Solution of the Resource-Constrained Activity Networks

by

Mohammed Moizuddin

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEMS ENGINEERING

June, 1996

SOLUTION OF THE RESOURCE-CONSTRAINED ACTIVITY NETWORKS

BY

Mohammed Moizuddin

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
In
SYSTEMS ENGINEERING

JUNE 1996

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by **Mohammed Moizuddin** under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE** in **SYSTEMS ENGINEERING**.

THESIS COMMITTEE

Dr. Shokri Z. Selim (Chairman)

Dr. Salih O. Duffuaa (Member)

Dr. Khaled S. Al – Sultan (Member)

Dr. Abdulbasit A. Andijani (Member)

Dr. Omar Al – Turki (Member)

Department Chairman

Dean, College of Graduate Studies

14 August, 96

Date

It is He who has

Created you from dust,

Then from a sperm-drop,

Then from a leech-like clot;

Then does He get you

Out (into the light)

As a child: then lets you

(Grow and) reach your age

Of full strength: then

Lets you become old, -

Though of you there are

Some who die before: -

And lets you reach

A term appointed:

In order that ye

May understand.

Quran, 40:67

Dedicated to

My

Mother,

Father,

and

Sisters

whose love, patience and perseverance

led to this accomplishment.

Acknowledgments

In the Name of Allah, Most Gracious, Most Merciful

Proclaim! (or read!) in the name of thy Lord and Cherisher, who created - Created man, out of a (mere) clot of congealed blood: Proclaim! And thy Lord is most bountiful, - He who taught (the use of) the pen, - Taught man that which he knew not. **Quran 96:1-5**

First of all, I thank *Allah(swt)* for being kind enough to give me all possible help and guidance, direct/indirect, without which this research would have remained a dream. Alhamdulillahi Rabbil Aalameen.

Next, I would like to thank King Fahd University of Petroleum and Minerals for providing the required support and resources in the completion of this research. I also thank Systems Engineering Department for the continuous support and encouragement.

I am grateful to my advisor, Dr. Shokri Z. Selim, for leading me in this area of research and for his consistent personal encouragement and support. I thank him for his priceless suggestions and time.

I am sincerely thankful to my committee members Dr. Salih O. Duffuaa, Dr. Khaled S. Al-Sultan, Dr. Abdulbasit A. Andijani and Dr. Umar Al-Turky for their valuable suggestions.

My appreciations are also due to all my friends who made my stay a memorable one. Their love erased home sickness from my dictionary.

Finally I thank my parents, sisters, and brother-in-law for their constant advice, love, sacrifice and prayers without which my graduation would have remained as an unconquered dream.

May Allah(swt) reward them all. Aameen.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Name: Mohammed Moizuddin

Title: Solution of the Resource-Constrained Activity Networks

Major Field: Systems Engineering

Date of Degree: June 1996

Resources available for the completion of each activity of a project (represented as an activity network) are always limited in the real life and so, finding the optimal schedule for the activities is very hard. Such a problem is termed as Resource-Constrained Project Scheduling (RCPS) Problem, which is classed as an NP-hard combinatorial optimization problem. Three solution methodologies based on simulated annealing, genetic algorithm and tabu search are developed. The proposed approaches are implemented on the 110 benchmark problems of Patterson. The parameters needed for each proposed algorithm were determined. Also, to validate the solution methodologies, a comparison is done with the popular procedures existing in the literature and the results are encouraging.

Master of Science Degree

King Fahd University of Petroleum and Minerals

Dhahran, Saudi Arabia

June, 1996

ملخص

الاسم : محمد معز الدين

العنوان : حل شبكات الاعمال في حالة الموارد المحدودة

المجال الرئيسي : هندسة النظم

تاريخ الدرجة : يونيو ١٩٩٦

نتطرق في هذا البحث الى جدولة شبكات الاعمال في حالة الموارد المحدودة. هذه المسائل تمثل الواقع أكثر من حالة افتراض توفر الموارد بلا حدود. غير أن هذه المسألة تصنف على أنها صعبة (NP-hard). ولقد استخدمنا ثلاثة مداخل لحل هذه المسألة. هذه المداخل تعتمد على طريقة محاكاة التلدين وطريقة البحث المحدود وطريقة علم الوراثة. وتم حل مجموعة مسائل باترسون المكونة من مائة وعشرة مسألة. وتم تحديد الثوابت اللازمة لكل طريقة. كما يضم البحث مقارنة مع أفضل طرق الحل المنشورة.

ماجستير العلوم

جامعة الملك فهد للبترول والمعادن

الظهران - المملكة العربية السعودية

يونيو ١٩٩٦

Chapter 1

Introduction

SUMMARY: This chapter gives an overview of the resource-constrained project scheduling problem, in general. The proposed research is also outlined and finally, the general assumptions made in this work are enumerated.

A project consists of a set of activities coordinated by technological precedence relationships/constraints which can be shown in a pictorial form. This pictorial representation is called as a project network or activity network. Any project network consists of two basic elements, *activities* (time consuming efforts) and *events* (signalling the start or end of activities) that are controlled by a *precedence* relationship.

Any project, large or small, requires a plan, a schedule, a system of control. We resort to a formal solution of finding a schedule by looking at the attributes of a project. The attributes of a project include (Willis [1]):

- a degree of complexity in the network,
- more than one of an important resource(s) is involved in the project, and

- the need to finish it within a certain time frame.

Generally, scheduling decisions are subject to both precedence constraints and resource constraints. Many researchers have dealt with a variety of situations in which one or both of these types of constraints are relaxed or simplified as we will see in the following sections of this chapter.

The organization of this chapter is as follows. In Section 1.1, a general classification of project scheduling problems will be presented. In Section 1.2, the resource-constrained project scheduling problem and its correspondence with assembly-line balancing and job-shop scheduling will be highlighted. In Section 1.3, the classification of RCPS problem will be predented. A formal 0-1 integer programming formulation of the concerned problem will be outlined in Section 1.4. The proposed research work will be enumerated in Section 1.5. Finally, in Section 1.6, the assumptions made in this work will be enumerated.

1.1 Classification of Project Scheduling Problems

Generally, to complete an activity of the project, resources are required. If the number of resources available are such that no activity starve for a resource or, in other words, the number of resources does not affect the start time of any activity, then such a project is termed as unlimited resource project. On the other hand, if the number of resources available affects the start time of any activity then such a project is termed as limited resource project. Thus, project scheduling literature can be classified under two main heads depending upon the availability of resources,as resource-unconstrained project scheduling and resource-constrained project schedul-

ing.

1.1.1 Resource-Unconstrained Project Scheduling

Project management is less difficult if only precedence relationships constrain the activity schedule. Two well known techiques under this category are the Critical Path Method (CPM) and the Project Evaluation and Review Technique (PERT). These two techniques were developed concurrently by the US navy for the polaris missile programme in the late 1950s. CPM and PERT are both digraphic longest path network models which require polynomial time computation (Ozdamar and Ulusoy [2]). The only difference that lies between CPM and PERT is the nature of the activity time. In CPM the activity time is deterministic and in PERT it is stochastic. Our work concentrates on CPM.

CPM is concerned with the scheduling of the many interrelated tasks that together form a complete project. Certain precedence constraints exist among the tasks and the problem is to complete the project as quickly as possible. Critical path problems take the form of a job-shop with an unlimited number of parallel machines, which is required to process a number of single-operation jobs between which precedence constraints exists (French [3]) and the aim is to minimize the make-span. This category of problems have very simple solution algorithm since the availability of machines does not limit the number of jobs that may be processed simultaneously.

1.1.2 Resource-Constrained Project Scheduling

In practice, activities do not get completed by their own; rather, they consume resources during their progress. Also, the resource availability (like workers, machines or money) is always limited. The scheduling problem becomes difficult to solve when the required resources are available in limited amounts, because the issue of allocating scarce resources among competing activities must be considered in optimizing a specific objective. This problem is known as the Resource-Constrained Project Scheduling (RCPS) problem and is NP-hard (Blazewicz et al. [4]). Coping with this problem is a theoretical challenge and an important contribution for practitioners. The following section gives a good overview of the problem at hand.

1.2 RCPS Problem

Consider the following example of a single project network, adapted from Hall et al. [5], for a good insight into the RCPS problem that is going to be considered. Figure 1.1 gives the activity network and Table 1.1 gives the information concerning activity duration and required resources.

Activity	Activity duration (days)	Required resource amounts (units)
A	3	4
B	6	3
C	4	5
D	5	2

Table 1.1: Activity duration and resource requirements for each activity.

If the amount of the resources available for this project is seven units or more, then the project duration will be 11 days (the longest path through the network

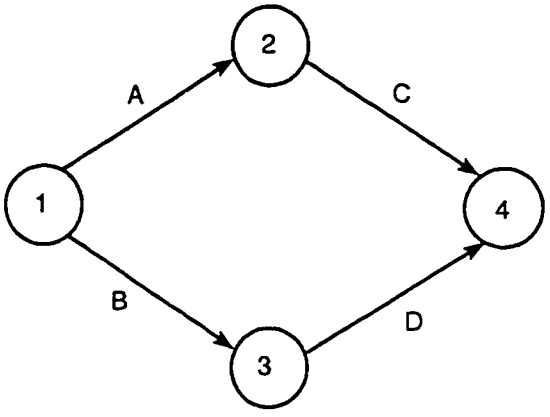

Figure 1.1: A simple project network.

as in CPM). In this case, the availability of resources is not a constraining factor. Let us suppose, however, that only six units of resources are available. In this case the minimum project duration will be 13 days. With five units of resources, the minimum project duration will be cut to 18 days. Thus, the amount of available resources are effecting the project duration.

The RCPS problem has attracted many researchers ever since the pioneering work of Kelley [6]. There have been many leaps made towards this problem to provide a schedule of activities that at no time collectively demand more than the resource availabilities. In general, the research in RCPS problem can be classified according to two criteria:

- objective function, and
- resource categories.

1.2.1 Popular Objective Functions

There are two popular objective functions used in RCPS research. The first one is the time-based objective of minimizing the project duration (e.g., Boctor [7]) and the second one is the cost-based objective of maxmizing the net present value (NPV) or minimizing the overall project cost (e.g., Yang et al. [8]). Our objective function is to minimize the project duration.

1.2.2 Resource Categories

Resources are classified under the following main heads: renewable, non-renewable, and doubly-constrained. If the resources are constrained on a period-by-period basis then they are termed as renewable resources. Like for instance, labour when used every day and limited on a daily basis. If the resources are constrained on a project basis then they are termed as non-renewable resources. Like for instance, the project budget can be considered as non-renewable resource if its total consumption over the whole project duration is limited to a certain value. Lastly, if the resources are simultaneously constrained on a period and project basis then they are termed as doubly constrained resources. Like for instance, if the cash consumed is limited on a daily basis and also for the overall project. Most of the published work deals with renewable resources (e.g., Leachman et al. [9], Sampson et al. [10]). Non-renewable resources were considered in Talbot [11], Patterson et al. [12], etc. Comparatively, very less amount of work has been done with doubly-constrained resources (Slowinski [13], Özdamar et al. [14]). The resource constraints complicate the representation of the problem and the more accurate they describe the actual problem, the more difficult they become to handle.

1.3 Classification of the Resource-Constrained Project Scheduling Problem

The RCPS problem can be classified into four distinct categories based on the number of projects being handled simultaneously and the number of resources being consumed by each project. Figure 1.2 briefly shows the classification.

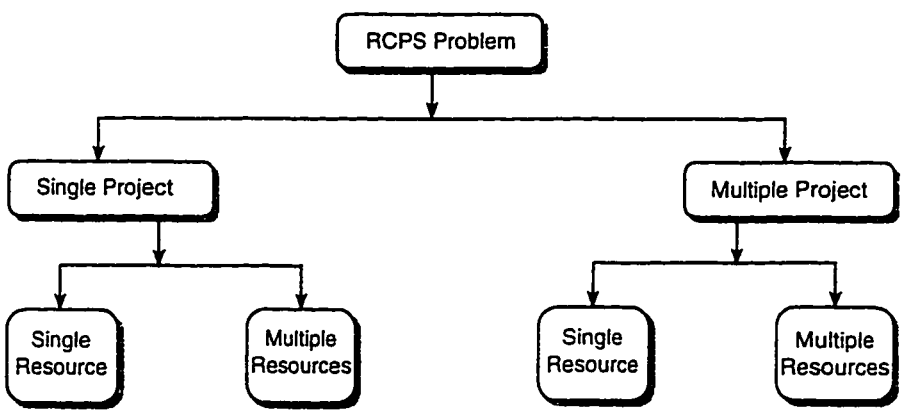

Figure 1.2: Classification of RCPS problem

Most of the reported research is concerned with single project single resource and single project multiple resource scheduling. Comparatively, a very less amount of work has been done for multiple projects scheduling (Dar-El et al. [15] and [16], Kurltus and Davis [17], Mohanty and Siddiq [18]). When multiple projects and multiple resources are considered then the problem becomes more tough because of the simultaneous solution of the problem relating to the timely completion of all the projects on one hand and effective utilization of resources on the other hand. Mohanty and Siddiq [18] mentioned the existence of enough scope for analyzing multiple projects-multiple resources constrained scheduling considering multi-objective optimization approach.

Assembly-line Balancing	Resource-Constrained Project Scheduling
Work elements	Activities
Work element times	Activity resource requirements
Work stations	Days
Cycle time	Maximum resource units available

Table 1.2: Relation between assembly-line balancing and RCPS.

1.4 Correspondence With Assembly-Line Balancing and Job-Shop Scheduling

Resource-constrained project scheduling problem has direct links with two well known problems of Industrial Engineering/Operations Research, which are assembly-line balancing problem and the job-shop scheduling problem. The following subsections highlights on these issues.

1.4.1 RCPS Problem Versus Assembly-Line Balancing Problem

Assembly-line balancing (ALB) problem, in general, can be stated as "the assignment of a set of tasks to a set of stations, while satisfying the precedence constriants, to minimize either the number of stations or the cycle time of the line". The former case is a type-1 ALB problem and the latter case is a type-2 ALB problem.

In project scheduling, as well, we have two abjectives. The first one is to minimize the project completion time under resource availability constriants, which is called as RCPS problem. The second one is to minimize the resources given a fixed project duration, which is called as resource levelling.

If we take a close look at the relationship depicted in Table 1.2 then we can conclude that minimizing of project completion time (in days) is equivalent to minimization of the number of stations (i.e., RCPS problem is equivalent to type-1 ALB problem) and minimization of available resources is equivalent to minimization of the cycle time of assembly lines (i.e., resource levelling is equivalent to type-2 ALB problem). This relation was first noticed by Moodie and Mandeville [19] and highlighted by Davis [20] and Andijani and Moizuddin [21].

1.4.2 RCPS Problem Versus Job-Shop Scheduling

Job-shop scheduling problem, in general, is stated as "the assignment of a number of jobs to a number of machines to optimize some performance measure (e.g., minimizing the makes span). If we consider a general precedence sturctured job-shop problem and replace machines by machine groups (for parallelism) which are required to process a number of single operation jobs, then the problem becomes a resource-constrained project scheduling problem. Davis [20] showed the conceptual similarities between the RCPS problem and the job-shop problem by an example.

1.5 0-1 Integer Programming Formulation of RCPS Problem

A cost minimization problem was formulated as a 0-1 integer program by Pritsker et al. [22]. Let us define variables

$$x_{jkt} = \begin{cases} 1, & \text{if job j is executed by resource k in time period t.} \\ 0, & \text{Otherwise.} \end{cases}$$

A binary (0-1) integer program is then formulated as:

Minimize

$$Z(X) = \sum_{j=1}^{J} \sum_{k=1}^{K} C_{jk} \sum_{t=ES_j+d_{jk}}^{LF_j} x_{jkt}$$

subject to

$$\sum_{k=1}^{K} \sum_{t=ES_j+d_{jk}}^{LF_j} x_{jkt} = 1 \quad \text{for all } j, \tag{1.1}$$

$$\sum_{k=1}^{K} \sum_{t=ES_j+d_{jk}}^{LF_j} t.x_{hkt} \leq \sum_{k=1}^{K} \sum_{t=ES_j+d_{jk}}^{LF_j} (t - d_{jk})x_{jkt} \quad \text{for all } j \text{ and } h \in V_j \tag{1.2}$$

$$\sum_{j=1}^{J} \sum_{q=t}^{t+d_{jk}-1} x_{jkq} \leq 1 \quad \text{for all k,t} \tag{1.3}$$

$$\sum_{j=1}^{J} \sum_{t=ES_j+d_{jk}}^{LF_j} d_{jk}.x_{jkt} \leq D_k \quad \text{for all k} \tag{1.4}$$

$$x_{jkt} \in \{0,1\} \quad \text{for all } j,k,t \tag{1.5}$$

Where

$Z(X)$ is the objective function value,

C_{jk} is the cost of performing job j with resource k,

J is the number of jobs,

K is the number of resources,

LF_j is the critical path latest finish time of job j,

ES_j is the critical path earliest start time of job j,

d_{jk} is the time required to perform job j with resource k,

V_j is the set of immediate predecessors of job j,

D_k is the total capacity of resource k.

The first set of constraints (1) requires that each activity is completed exactly once. The second set of constraints (2) represents the precedence relations between activities, i.e., the precedence relations are maintained. Constraints (3) and (4) represent per period and total availability resource constraints, respectively. This means that the resource allocated to activities at any time during the project duration do not exceed the resource availibility. The last constraint (5) represents the binary variable x_{jkt}.

The assumptions made in the above model are:

1. Preemption of any activity is not allowed, i.e., once an activity is started, it can not be interrupted;

2. Each activity has a known, deterministic, integer duration;

3. Each activity consumes known, deterministic level of resource of each kind during its processing time;

4. The level of availability of each resource type is assumed to be constant throughout the schedule.

1.6 Objectives

The objectives of this thesis research are enumerated as follows.

- To develop new algorithms based on Simulated Annealing, Genetic Algorithm, and Tabu Search and implement these algorithms on the 110 standard benchmark problems of Patterson.
- To conduct the parametric studies of these developed algorithms.
- To compare the performance of the above mentioned algorithms with the best heuristics developed so far using the 110 standard test problems of Patterson.
- To do the difficulty analysis of the 110 standard problems of Patterson.

1.7 Assumptions

The general assumptions made in achieving the above goals are as follows.

- Each activity has a known integer duration.
- No preemption is allowed.
- Each activity requires known level of renewable resources of *multiple* kind.
- The level of availability of the resources of multiple types are constant throughout the schedule.
- The on-hand resources will be used for the activities of the *single project* network under consideration.

Chapter 2

Literature Survey

SUMMARY: In this chapter a brief literature review on resource-constrained project scheduling problem is done. Existing approaches (both, exact and inexact approaches), the so called efficient approaches, and existing global procedures are discussed.

The literature on resource-constrained project scheduling problem is very rich. Numerous heuristics and several exact solution techniques have emerged. The first complete survey on RCPS was done by Davis [20] followed by Herroelen [23], Willis [1], Icmeli et al. [24] and Ozdamar and Ulusoy [2].

The primary concern of this survey was the project scheduling problems with fixed resource limits and renewable resources. This chapter is organised as follows. In the following section we deal with the existing literature on RCPS problem. In Section 2, the popular approaches/heuristics will be dealt. Finally, in Section 3, the existing global optimum technique is dealt with.

2.1 Existing Approaches

The literature contains exact and heuristics methods. As mentioned in chapter 1 that the objective functions used are classified into cost-based objectives and the time-based objectives. Table 2.1 briefly shows the work done using both of these objective functions. We will concentrate on the research done on time-based objectives.

Table 2.1: RCPS research classification

Time based objective	Cost based objective
Alvarez & Tamarit [25]	Baroum & Patterson [26]
Bell & Park [27]	Doersh & Patterson [28]
Bell & Han [29]	Elmaghraby & Herroelen [30]
Boctor [31]	Russell [32]
Boctor [7]	Russell [33]
Christofides et al. [34]	Smith & Aquilano [35]
Cooper [36]	Yang et al. [8]
Davis & Heidorn [37]	
Davis & Patterson [38]	
Khattab & Choobineh [39]	
Khattab & Choobineh [40]	
Patterson & Huber [41]	
Patterson & Roth [42]	
Pritsker et al. [22]	
Shrage [43]	
Stinson et al. [44]	
Talbot & Patterson [45]	
Talbot [46]	
Thesen [47]	
Weglarz [48]	
Willis & Hastings [49]	

Numerous heuristic procedures and exact solutions have emerged considering

the objective of minimizing the project completion subject to renewable resource constriants.

2.1.1 Exact Procedures

Pritsker et al. [22] formulated the multi-resource multi-project scheduling problem as a 0-1 integer programme. Later, improvements in this formulation were done by Patterson and Huber [41] and suggested two bounding algorithms that examine the feasibility of a series of 0-1 programming problems. Another improvement was done by Patterson and Roth [42], followed by Talbot and Patterson [45]. Taveres [50] proposed a flexible mathematical formulation that avoids large numbers of binary variables. In this approach he represented a project as a sequence of stages where the duration of every stage is treated as a decision variable and the constriants were linear.

Davis and Heidorn [37] formulated the multiple resource-constrained problem as an integer programming problem and developed a branch and bound algorithm for solving it. The algorithm they developed is similar to Johnson's [51] branch and bound algorithm with differences in the node selection heuristics employed. Talbot and Patterson [45] described another integer programming algorithm with network cuts to solve the general resource-constrained scheduling problem. Patterson [52] evaluated the procedures due to Stinson et al. [44] and Talbot and Patterson [45] with respect to computer storage, solution time and the number of problems solved optimally within a reasonable computation time, the results indicating that Stinson's branch and bound procedure is best when computer memory is not the limiting factor.

The abundance of zero-one variables and the large number of constraints have led researchers to develop branch and bound procedures for the resource-constrained project scheduling problem. The contributions made in this field are by Christofides et al [34], Hastings [53], Shrage [54], Stinson et al. [44], Willis and Hastings [49]. The computational success of a branch and bound algorithm depends on its branching technique and the strength of its lower bound. Problem-specific branching techniques were proposed by Bell and Park [27], Christofides et al. [34], Demeulemeester and Herroelen [55].

Ozdamar and Ulusoy [2] justified that the branching scheme found in Bell and Park [27] is suitable for loose-moderate tightness of resource-constraints, whereas that found in Demeulemeester and Herroelen [55] is specifically powerful in tightly constrained problems. The effective lower bounds, dominance rules and branching techniques applied in the branch and bound procedures are observed to be insufficient for problems of practical size, however efficient they are (Ozdamar and Ulusoy [2]).

2.1.2 Heuristic Procedures

The need to solve problems of practical size has motivated researchers to develop effective heuristics. It was initiated by Weist [56] and by the late 1970s, there were already more than 100 heuristic procedures (e.g., Bell and Hans [29], Boctor [7, 31], Cooper [36], Khattab and Choobineh [40], Thesen [57]).

Davis and Patterson [38] and Cooper [36] classified heuristics into two types: serial heuristics, where activity priority is predetermined and remains fixed throughout the scheduling procedure, and parallel heuristics, where activity priority is updated

each time an activity is scheduled. Efforts were made by Davis and Heidorn [37], Patterson [58] and Thesen [57] to identify problem characteristics. The effect of problem characteristics on dispatching rules were studied by Patterson [58] and Ulusoy and Ozdamar [59].

A strategy for using heuristics is given by Boctor [31], who proposed to run the problem each time with another well-reputed heuristic rule and keep the best solution obtained. Boctor also provided probabilities for obtaining optimal results for specific rule combinations. Bell and Hans [29] and Ulusoy and Ozsamar [60] developed heuristic procedures in which problem dependency is eliminated.

In general, the basic structure of most heuristic procedures for RCPS problem is to use a priority rule to rank the activities and schedule the ranked activities such that the resource limitations are not violated. The most unfortunate thing is that no heuristic has been found to perform well for a wide variety of project network topologies and resource levels. The only concentration in developing them was minimization of the computational time.

2.2 Popular Heuristic Procedures

In a recent survey by Ozdamar and Ulusoy [2] it was found that the most popular and efficient heuristics that exists in the literature are five, as enumerated below.

- The heuristic procedure due to Demeulemeester and Herroelein [55],
- WRUP (weighted resource utilization and precedence) by Ulusoy and Ozdamar [59],
- LCBA (local constraint based analysis) by Ulusoy and Ozdamar [60],

- LI-WI by Li and Willis [61], and
- the heuristic procedure due to Bell and Han [29].

  Out of these, for the first heuristic (Demeulemeester and Herroelein), only the solution time is reported. But they are not comparable owing to the differences in operating systems, compiler, coding and computer power.

2.3 Genetic Algorithms Developed For RCPS

Two research papers dealing with the same genetic algorithms were developed by Cheng and Gen [62, 63]. They designed a crossover operator and a mutation operator. The crossover operator designed by them to perform wide spread search and the mutation operator to perform an intensive search. They concentrated on project duration only and no computational time was reported. They took two benchmark problems consisting of 27 activities and three types of resources each, and compared their optimum project durations with that of 8 heuristics. They also gave the information regarding the generation in which they got the optima.

Chapter 3

Preliminary Network Operations

SUMMARY: Storing the network is an important issue which is shown here. The approach of storing the network is similar to the array-based implementation of singly linked list.

The RCPS problem will be represented as a network. The nodes of the network will represent the activities while the arcs represent the precedence relationships among the activities. This give rise to an activity-on-node (A-on-N) network. Preliminary network operations like storing the network in the computer, labelling the network in a uniform manner and finding the ancestors of each node are very much useful to reduce the computational time. This section is dedicated to these three issues.

3.1 Data Structure of the A-on-N network

Over the years, analysts have studied different ways to store and manipulate network data within the computer memory. A common techinque of storing the network is by *node-arc incidence matrix* (a two dimensional array) in which nodes are represented as rows and arcs are represented as columns (Ahuja et al. [64]). A characteristic of this matrix is that each column has exactly two non-zero entries. one being a +1 and the other a -1. Thus, if we have N nodes and A arcs then the number of non-zero entries will be $2A$ and the number of zeros will be $NA - 2A$. For a small network as shown in Figure 3.1, the node-arc incidence matrix will contain 24 non-zero entries and 96 zero entries. Thus node-arc incidence matrix is a sparse matrix and is an inefficient way to store in the computer.

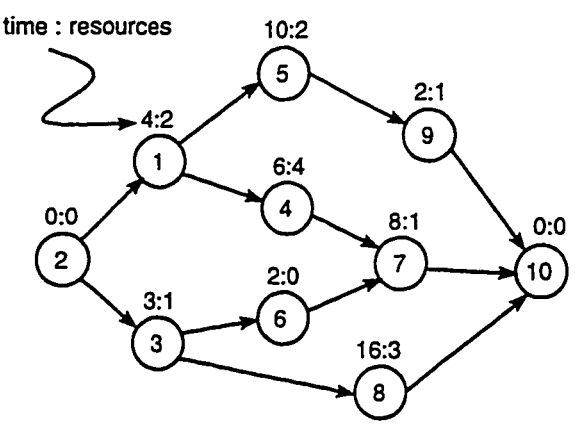

Figure 3.1: Example network with node number inside the circle

We used singly linked list (Ahuja et al. [64]) to store the network by defining two single dimensional arrays *start* and *pred* of size $N + 1$ and $A + 1$ respectively. These two arrays define $N + 1$ and $A + 1$ cells respectively, indexed from 1 through $N + 1$ and 1 through $A + 1$. Array *start* contains indices which refer to the position

in array *pred*. Each node has a list which contain all its immediate predecessors. The lists are concatenated and formed an array *pred*. To clarify, consider node i, element $start(i)$ contains an index say S_i. The element $pred(S_i)$ in the array *pred* is the start of the list of predecessors of node i. To retrieve all the predecessors of i from *pred* one has to obtain the indices $start(i)$ and $start(i+1)$, S_i and S_{i+1}. then the list of predecessors is $pred(S_i), pred(S_{i+1}), \ldots\ldots, pred(S_{i+1}-1)$.

The predecessor of source node is taken as zero. To store the activity time and the resource requirement for each activity (represented as node), two more single dimensional arrays, *actime* and *res*, of size N are used.

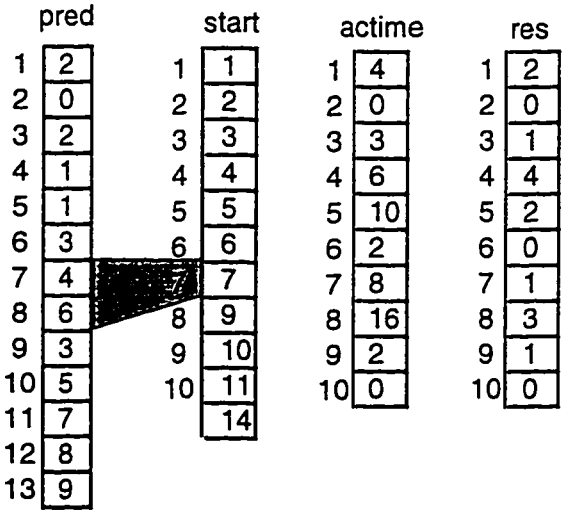

Figure 3.2: Linked list representation of the example network.

The above approach is applied on the example network shown in Figure 3.1 and the resulting arrays are shown in Figure 3.2. If one needs to find the predecessors of, say, node 7 (the shaded area in Figure 3.2) then from the *start* array we find the position of the first predecessors of node 7 ($start(7) = 7$) and node 8 ($start(8) = 9$) in the *pred* array. We are using the very next node (i.e., node 8) because the end

of the predecessor list of node 7 in the *pred* array will be one position less than the first predecessor of node 8. Thus,

Predecessors of node 7 = $\{pred(k)$ such that $start(7) \leq k < start(8)\}$

$= \{pred(7), pred(8)\} = \{4, 6\}$

3.2 The Labelling Algorithm

On any activity-on-node (A-on-N) network, nodes are differentiated from one another by a unique numerical value assigned to each node. This numerical value is called the *label* of the node. Generally, when the network becomes complex, manual labelling of the nodes could be random or non-sequential, i.e., the label of any node could be less than the label of its predecessor(s) or greater than the label of its successor(s). Non-sequential labelling makes the analysis of the network a complex issue.

we propose the `labelling` algorithm which is basically used to convert the randomly labelled nodes of any A-on-N network into sequentially labelled nodes, where any node label in the network will have a numerical value which is always greater than its immediate predecessor(s) and less than its immediate successor(s).

Algorithm 1 gives a formal algorithmic description of the `labelling` algorithm. In this algorithm we first search for the source node and label it as one. Next we search for a node which has all its predecessors already labelled. Then we label this node with the next available label. This procedure is repeated until all nodes are labelled.

Table 3.1 illustrate the `labelling` algorithm applied to the example network. Figure 3.3 shows the example network given in Figure 3.1 after it has been labelled

```
algorithm labelling;
begin
  Let Ω = {1,2,...,N} be a list of unlabelled nodes;
  Check for a node, k, with zero predecessor and label it as 1;
  Ω := Ω\k;
  initialize label := 2;
  while Ω ≠ Φ do
  begin
   Find a node k ∈ Ω with all its predecessors labelled;
   label of the node k is label;
   Ω := Ω\k;
   label := label + 1;
  end while;
end labelling;
```

Algorithm 1: Pseudo code for `labelling` algorithm.

as above.

3.3 An Algorithm for getting the ancestors of a node

Our approach for the RCPS problem requires the anscestors of each node. as will be seen later. The pseudo-code shown in the Algorithm 2. creates the ancestors for each node of the network. It should be noted. however. that the source node does not have any ancestor (or the ancestor of the source node is zero) and the sink node (N) has all the N-1 nodes as its ancestors.

The algorithm works as follows. First we assign a value zero to the ancestor of the source node (labelled as 1 by the `labelling` algorithm). Then for each of the next $N-1$ nodes the algorithm proceeds by taking the labels of predecessors of that node and their ancestors.

Table 3.1: Implementation of `labelling` algorithm on the example network

Iteration	Ω	Selected Node	*label*
0	1,2,3,4,5,6,7,8,9,10	2	1
1	1,3,4,5,6,7,8,9,10	3	2
2	1,4,5,6,7,8,9,10	6	3
3	1,4,5,7,8,9,10	8	4
4	1,4,5,7,9,10	1	5
5	4,5,7,9,10	4	6
6	5,7,9,10	5	7
7	7,9,10	7	8
8	9,10	9	9
9	10	10	10

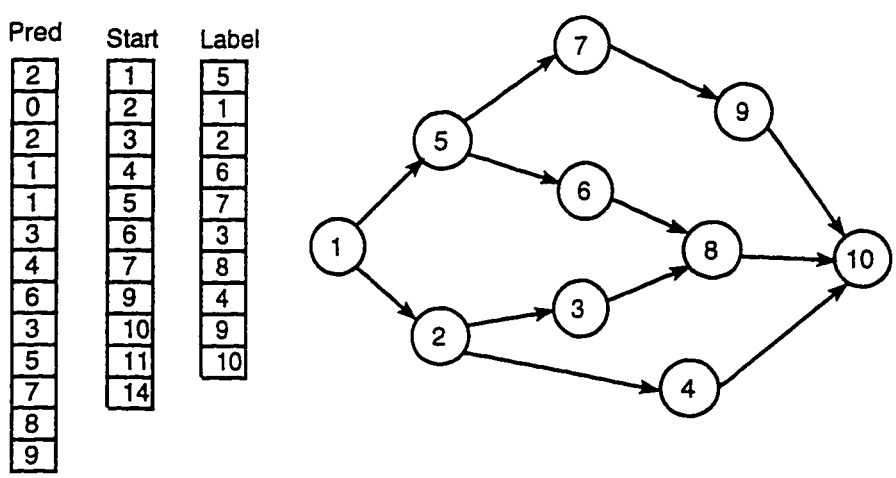

Figure 3.3: Nodes labelled by the labelling algorithm

```
algorithm ancestors;
begin
 set AncestorList(1) := 0;
 for nodes i := 2,3,4,...,N − 1 do;
 begin
  set AncestorList(i) := {};
  for predecessor of j (where S_i ≤ j < S_{i+1}) of node i do;
   AncestorList(i) := AncestorList(i) ∪ AncestorList(j);
 end for;
end;
```

Algorithm 2: Pseudo code for ancestors algorithm

Chapter 4

Simulated Annealing Algorithm

SUMMARY: A brief background of simulated annealing is given. A new simulated annealing algorithm for the RCPS problem is proposed.

We propose a two-stage simulated annealing algorithm for the single project scheduling problem when the resources of multiple types considered are renewable from period to period. The algorithm works by switching the priorities of two parallel activities, randomly and calculating the critical path.

4.1 Simulated Annealing

Kirkpatrick, Gelatt and Vecchi [65] and Cerny [66] introduced a powerful tool for combinatorial optimization problems called Simulated Annealing (SA). SA has been used to solve a variety of combinatorial optimization problems like the clustering problem (Selim and Al-Sultan [67], Al-Sultan and Selim [68]), VLSI design (Sait and Youssef [72, 73]), the job shop scheduling problem (French [3]), etc. For more

on the applications of SA refer to Otten et al. [69], Eglese [70] and Koulamas et al. [71].

Simulated annealing is a random search technique which imitates the physical annealing of solids. The process of annealing begins with a material in a high energy state (high temperature), and then lowering its temperature gradually, analogous to decreasing an objective function value (usually referred to as the cost) by a series of moves. The main advantage of SA over descent algorithms is that it attempts to avoid being trapped in a local optima by sometimes accepting uphill moves also.

The probability of accepting an uphill move is given by

$$P_a = e^{-\delta/T}$$

where δ is given as the change in objective function value and T is the control parameter which corresponds to temperature in physical annealing. If $\delta < 0$ (i.e., improvement in objective function value) then the move is automatically accepted. Otherwise an uphill move is accepted with a given probability, P_a.

The temperature, T, is reduced after a certain fixed number (called the chain length, L) of configuration changes are made. With the decrease in temperature the probability of accepting an uphill move decreases. The procedure is stopped at a point (temperature) where P_a is almost zero (or some suitable stopping criteria). This point is called freezing point or freezing temperature. The pseudo-code of a simple SA algorithm is given in Algorithm 3.

From the analogy, the different states of the substance correspond to the different feasible solutions to the combinatorial optimization problem, and the energy of the system corresponds to the objective function to be minimized (Kirkpatrick et al.

```
algorithm SA;
begin
 generate an initial solution S_c with an objective function
 value f_c;
 set T := T(0);
 while T ≥ T_f do
 begin
  set chain length counter CL := 0;
  while CL ≤ L do;
  begin
   generate a neighbour S_t of S_c with an objective function
   value f_t;
   δ := f_t − f_c;
   if δ ≤ 0 then S_c = S_t;
   else
   begin
    generate a random number u;
    if u < P_a then S_t = S_c;
   end else;
   CL := CL + 1;
  end while;
  decrease T;
 end while;
end SA;
```

Algorithm 3: A Pseudo code of a simple annealing algorithm.

[65]). The following generic choices (called the cooling schedule) must be made for the implementation of SA:

1. the initial temperature, $T(0)$,
2. a temperature function, $T(t)$, to decrease the temperature,
3. the chain length (L), and
4. the freezing temperature (T_f).

The proposed simulated annealing algorithm for the RCPS problem with single project and multiple types of renewable resources is discussed in the next section.

4.2 Application of SA to the RCPS Problem

The application of simulated annealing to the resource-constrained project scheduling problem is shown in pseudo code in Algorithm 4. The program starts labelling all the nodes by using the algorithm `labelling` discussed earlier in Section 3.1.2. The main purpose of this `labelling` algorithm is to efficiently search the predecessors, successors or ancestors of any node of the project network.

The next step is to update the whole network as per the new node labels obtained from the `labelling` algorithm. Get the ancestors of each node in the network by using the algorithm `ancestors` as discussed in Section 3.3. These ancestors are then used to assign priorities to the activities using algorithm `priority` discussed in Section 4.2.1. These priorities are used to schedule the activities and determine the corresponding project duration which is stored as current critical path (CP_C). In the following subsections we discuss the algorithms for generating initial priority

```
algorithm annealing;
begin
 label all the nodes by algorithm labelling;
 update all the lists as per the new labels;
 get the ancestors of each node by algorithm ancestors;
 generate an initial priority assignment, A_c with critical path
 CP_c by algorithm priority;
 determine the project duration, CP_c, by algorithm critical path;
 get the optimum chain length, L_opt, by algorithm stage1;
 set T := T(0);
 while T ≥ 1/ln(1/ε) or CP_c remains constant for 15 iterations
 at any one level do;
 begin
  decrease the temperature according to temperature decrement
  function;
  for CL := 1,2,...,L_opt do
  begin
   generate a neighbour A_t of A_c by algorithm switching;
   determine the project duration, CP_t;
   if CP_t ≤ CP_c or U(0,1) ≤ P_a(= e^((CP_c - CP_t)/T)) then CP_c := CP_t
  end for
 end while
end annealing;
```

Algorithm 4: Pseudo code for Simulated Annealing algorithm.

assignment and evaluating the corresponding project duration. Also, the two annealing stages are discussed with the neighbour generating technique at the end of this section.

4.2.1 An Algorithm for Generating an Initial Priority Assignment

```
algorithm priority;
begin
  set Ψ = {1} and Γ = Φ;
  while n[Ψ] < N do
  begin
    set node := 2;
    while node ≤ N do
    begin
      if node ∉ Ψ and each element of AncestorList(node) ∈ Ψ then
      begin
        Γ := Γ ∪ {node};
        node := node + 1;
      end if
      else node := node + 1;
    end while;
    generate U_k = U(0,1) ∀k ∈ Γ;
    let U_max = max{U_k : k ∈ Γ} and corresponding k be k_max;
    Ψ = Ψ ∪ {k_max};
    Γ = Γ\{k_max};
  end while
end priority
```

Algorithm 5: Pseudo code for priority algorithm

The algorithm priority, whose psuedo code is given in Algorithm 5, starts by giving the highest priority to the source node (node 1). It then checks how many nodes are available for assigning a priority. I.e., it checks the nodes with all predecessors

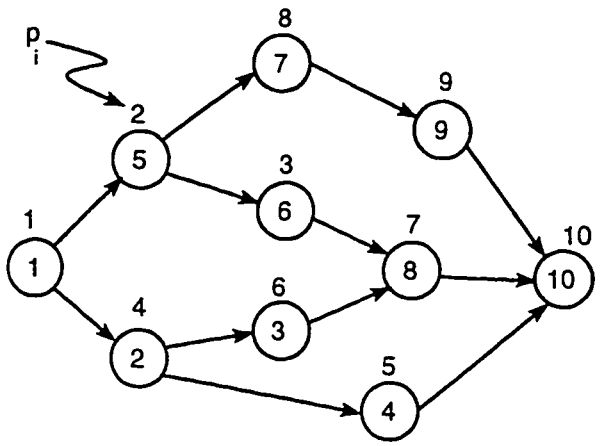

Figure 4.1: Prioritized network

already prioritized. It then assigns a random number for each such node and the node with largest random number will be assigned the next priority. The procedure is repeated until all the nodes are allotted a unique priority. In the pseudo-code the set Ψ is an ordered set. It contains the indices of nodes in their order of priority. Figure 4.1 shows the example network with priority allotted to each node by the `priority` algorithm. Note that the priority of the predecessors of node i are less than p_i and the priority of the successors of node i are greater than p_i. This condition is termed as the *uniformity criteria* and any network satisfying this condition is termed as a *uniform network.*

4.2.2 An Algorithm for Evaluating a Prioritized Network

Getting the minimum critical path of the multiple resource-constrained project network is the objective of this research. Every activity requires a deterministic number of resources (of multiple types) for its completion. If the required number of resources, is not available then that activity is delayed until at least all the required

```
algorithm critical path;
begin
    set TSTART_1 = 0, TFIN_1 = t_1 and TNOW = t_1;
    let Π = {1} be a set of Scheduled Nodes, and Λ := {} be a set of
    those nodes whose all predecessors are in Π;
    while n[Π] < N do
    begin
      for i := 2,3,....,N if i ∉ Π and all predecessors of i ∈ Π
      then Λ := Λ ∪ {i};
      for all nodes in Λ do
      begin
        let l ∈ Λ be the node with highest priority among all
        nodes in Λ;
        if R_lk ≤ netres(k) ∀k := 1,2,...,NTR then
        begin
          set TSTART_l := TNOW and TFIN_l := TSTART_l + t_l;
        end if;
        end for :
      TNOW := min{TFIN_m ∀m ∈ Π such that TFIN_m > TNOW}, and
      let h be the corresponding node;
      netres(k) := netres(k) + R_hk ∀  k;
    end while;
end critical path;
```

Algorithm 6: Pseudo code for `critical path` algorithm.

number of resources of all types become free and preceeding activities are completed. This analysis is repeated for all the activities which constitute the project. The psuedo-code for the critical path algorithm for any resource-constrained project network with resources of multiple types is shown in Algorithm 6.

Let $TSTART_l$ be the actual start time of activity l if the resources, time and precedence restrictions are taken into account,

t_l be the duration of activity l,

$TFIN_l$ be the completion time of activity l,

$TNOW$ be the time at which assignments are now being considered,

R_{ik} be the required number of resource(s) of type k for activity i,

NTR be the number of resource types,

netres(k) be the net resources of type k available at time $TNOW$. At $TNOW = 0$, *netres* is the amount of resources available for project.

The algorithm works as follows. First consider the source node, set its $TNOW$, start-time, $TSTART_i$, and finish time, $TFIN_i$, to zero i.e., $TSTART_1 = 0$ and $TFIN_1 = 0$. We permanently mark this scheduled node (or activity). Next, the algorithm searches for those activities whose all predecessors are permanently marked. Such activities are then temporarily marked. Out of these temporarily marked activities, the algorithm selects that activity which has the highest priority. The algorithm then checks whether the resources, of all types, required for the completion of this selected activity is less than or equal to the net available resources (of all types). If the condition is not satisfied then it takes the node with the next highest priority, among the temporary marked nodes. And if the condition is satisfied then it schedules the activity and calculates its $TSTART_i$ ($= TNOW$) and $TFIN_i$ ($= TSTART_i + t_i$). The procedure continues till the sink node is scheduled. $TFIN$ of the sink node is the finish time of the project, i.e., the length of the critical path for a particular sequence of activities.

4.2.3 Annealing Stages

The annealing steps could be described in two stages. The first stage determines the optimal chain length, L_{opt}, and the second stage performs the annealing using this L_{opt} and other annealing parameters such as initial temperature, temperature

decrement function and stopping criteria as discussed in the next section. These two stages are described in the following subsections.

Stage 1

The number of neighbours generated at each temperature level is called the chain length, L. The value of L should be large so that the system attains equilibrium at each temperature level (Kirkpatrick et al. [65]). In this study the chain length at each temperature level will be kept constant. But, how large should be the value of L?

```
algorithm stage1;
begin
 initialize ap, z1 and z2 to zero and L to Lo;
 while ap ≤ δa do
 begin
  for C := 1,2,...,L do
  begin
   z1 := z1 + 1;
   generate a neighbour and let its critical path be CPt;
   if CPt ≤ CPc or U(0,1) ≤ e^((CPc - CPt)/T(0)) then set CPc := CPt and
   z2 := z2 + 1;
  end for
  ap := 100(z2/z1) and set L := L + Δ;
 end while
 Lopt := L - Δ;
end stage-1;
```

Algorithm 7: Pseudo code for getting the optimum chain length (stage 1)

The chain length in this study is selected such that at the initial temperature, $T(0)$, the acceptance rate of neighbours is at least δ_a percent. This is an attempt to bring the system to a state corresponding to thermal equilibrium in the physical analog. Hence, we propose a search algorithm to get the optimal value of chain

length, L_{opt}, which will remain constant at every temperature level. The pseudo-code is given in Algorithm 7.

The algorithm makes a move (by generating a neighbour) and checks whether the move is accepted or not. After repeating this L times, it then checks whether the percentage of acceptance (ap) is greater than δ_a. If yes, then the current value of L is L_{opt}. Else, the value of L is incremented by Δ and the above procedure is repeated.

Stage 2

The output of stage 1, L_{opt}, is the input to Stage 2. In this stage the temperature is decreased in discrete steps according to the temperature function, $T(t)$. At each temperature level L_{opt} feasible neighbours are generated and the critical path of each neighbour is stored as temporary critical path, CP_t. This neighbour is accepted if either $CP_t \leq CP_c$ or $e^{\frac{CP_c - CP_t}{T}} \geq U(0,1)$ where CP_c is the critical path of the current priority assignment. Acceptance of a neighbour means that it replace the current priority assignment and the next priority assignment will be a neighbour of newly accepted neighbour.

The selection of the parameter values of the above two stages are most important and are highlighted in Section 4.3. But first we discuss the neighbour generating technique which is taken in the following subsection.

4.2.4 An Algorithm for Generating a Feasible Neighbour

Any two nodes, i and j, in a uniform network are said to be parallel if $\min\{i,j\}$ does not lie in the ancestor's list of $\max\{i,j\}$. In the example network nodes 4 and 6 are

parallel while nodes 2 and 8 are not. Algorithm 8 shows an algorithm for finding two parallel activities. The priorities of nodes i and j, represented by p_i and p_j, can be switched and the resulting critical path of the resource-constrained network is calculated. After switiching the priorities the uniformity of the network should be maintained.

```
algorithm parallel-activities;
begin
 Generate two U(1,N) integer random numbers, U1 and U2;
 Na := min{U1,U2} and Nb := max{U1,U2};
 if Na ∈ Ancestor(Nb) then start again;
 return Na and Nb;
end;
```

Algorithm 8: Pseudo code for the algorithm `parallel-activities`

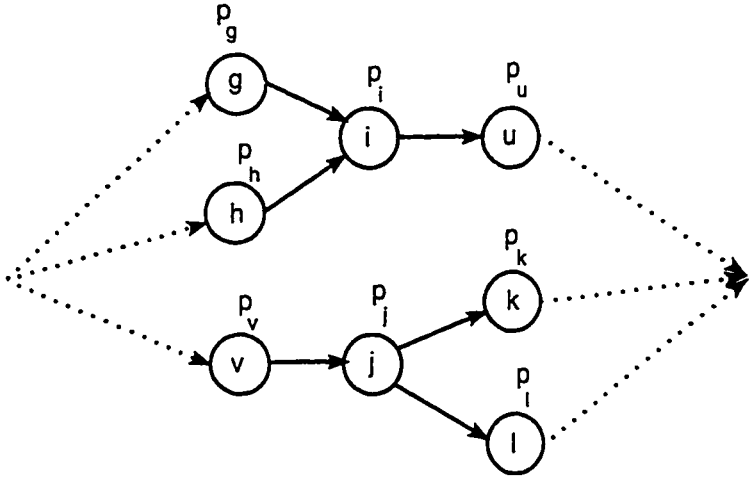

Figure 4.2: Network showing the parallel activities i and j

Let us consider the switching of the priorities of two randomly selected nodes, i and j, such that $1 < i < N, 1 < j < N$ and i, j are parallel, as shown in the Figure

4.2 . Note that the source node and the sink node are not parallel to any node in the network and so their priorities will always remain as 1 and N, respectively. But the priorities of the remaining $N-2$ nodes may change. Let us assume that the network shown in Figure 4.2 satisfies the uniformity criteria, i.e., $p_g, p_h < p_i < p_u$ and $p_v < p_j < p_k, p_l$

Let us also assume that $p_i > p_j$. Let $p_{min} = p_j$ and $p_{max} = p_i$. After switching the priorities of nodes i and j we observe that since $p_i > p_j$, then $p_j < p_u$ and $p_i > p_v$. In other words, p_{min} will not effect the forward star at node i and p_{max} will not effect the backward star of the network at node j. But p_{min} might effect the backward star at node i and p_{max} might effect the forward star at node j. Thus the following two cases might arise at these two nodes:

At node i:

Case(i) - if $p_j > p_g$ and p_h then the uniformity criteria is still maintained.

Case(ii) - if $p_j < p_g$ or p_h then switch p_j with $max\{p_g, p_h\}$ which will maintain the uniformity criteria, between node i and its immediate predecessors.

At node j:

Case(i) - if $p_i < p_k, p_l$ then the uniformity criteria still maintained.

Case(ii) - if $p_i > p_k$ or p_l then switch p_i with $min\{p_k, p_l\}$ which will maintain the uniformity criteria, between node i and its immediate successors.

The switching will continue between successive nodes till the uniformity criteria is reached for the whole network.

The pseudo-code given in Algorithm 9 gives the details of the above approach utilized in the `switching` algorithm. The algorithm works as follows. Switch the priorities of the two parallel activities selected by algorithm `parallel-activities`

```
algorithm switching;
begin
 select two parallel nodes N_a and N_b with priorities p_a and p_b
 (p_a < p_b) using algorithm parallel − activities;
 switch the positions of the priorities p_a and p_b;
 while p_a <priorities of immediate predecessor of N_b do
 begin
  MAX := maximum priority among all the immediate predecessors
  of N_b;
  set N_MAX to be the node corresponding to MAX;
  switch the position of the priorites p_a and MAX;
  set N_b := N_MAX;
 end while;
 while p_b > priorities of immediate successors of N_a do
 begin
  MIN := minimum priority among all the immediate successors
  of N_a;
  set N_MIN to be the node corresponding to MIN;
  switch the position of the priorities of p_b and MIN;
  N_a := N_MIN;
 end while;
end;
```

Algorithm 9: Pseudo code for switching algorithm

and check whether the network is still uniform at these two nodes.

1. If the network is not uniform at the node with priority p_{min} then we take the maximum priority of its predecessors and switch it with p_{min}. This is continued till the uniformity criteria is satisfied.

2. If the network is not uniform at the node with priority p_{max} then we take the minimum priority of its immediate successors and switch it with p_{max}. This step is continued till the network becomes uniform.

Let us consider the prioritized network in Figure 4.1. Switch the priorities of nodes 4 and 8. Note that $p_4 = 5$ and $p_8 = 7$. Hence,

$$p_{min} = min\{5, 7\} = 5,$$

$$p_{max} = max\{5, 7\} = 7$$

p_{max} will not effect the uniformity criteria at node 4. But p_{min} effects the criteria at node 8. To satisfy the uniformity criteria we move in the backward star direction by choosing the predecessor of node 8 with highest priority, i.e., node 3 with priority 6. Now, the priorities of nodes 8 and 6 are switched and thus the network becomes uniform.

4.3 Parameters Selection

Parameters selection of simulated annealing plays a vital role in determining the performance of the algorithm and constitute the annealing or cooling schedule. For the proposed SA algorithm the following choices need to be made:

- the initial chain length, L_o,
- the value of δ_a, and the increment, Δ.
- the initial temperature, $T(0)$,
- the temperature function, $T(k)$, and
- a stopping criteria to terminate the algorithm.

4.3.1 The Initial Chain Length, L_o

The initial chain length, L_o, should be large. But, even a low value of L_o will also work because Stage 1 of the proposed algorithm will increase the chain length from L_o with an incremental value, Δ, if the acceptance percentage, ap, at temperature, $T(0)$ is less than or equal to δ_a. These increments in chain length will be stopped if $ap > \delta_a$.

4.3.2 The Value of δ_a

The value of δ_a should be as high as possible because initially one would like to explore as many priority assignments as possible by accepting $\delta_a\%$ moves.

4.3.3 The Value of Δ

The increment, Δ, in the chain length (L) is taken when $ap \leq \delta_a$. To reach a value of L where $ap > \delta_a$, large strides are to be taken.

4.3.4 The Initial Temperature, $T(0)$

As mentioned earlier, the probability of accepting a priority assignment with higher critical path than that of the current assignment is given as $P_a = e^{-(CP_t - CP_c)/T}$. If T is large then P_a will also be large. Initially at $T = T(0)$ one would like to explore the priority space by accepting almost all the moves. By setting $T(0)$ to be large enough, virtually all transitions are accepted which corresponds to heating up a substance to a high temperature (molten state) until all the particles are randomly arranged in the liquid or until all possible changes in the structure are expected.

4.3.5 The Temperature Function, $T(k)$

Temperature decrement or annealing should be done slowly, otherwise the resulting solid will not attain the ground state, but will be frozen into a metastable state, such as a glass or a crystal with several defects in the structure (Kirkpatrick et al. [65]). In optimization this corresponds to stopping at non-optimal point.

Laarhoven and Aarts [74] proposed a temperature function which proved successful and robust under empirical testing. Their proposed function gives the temperature $T(k+1)$ at iteration $k+1$ as

$$T(k+1) = \frac{T(k)}{1 + \frac{T(k)\ln(1+\delta)}{3\sigma_{T(k)}}} \tag{4.1}$$

where $\sigma_{T(k)}$ is the standard deviation of the objective function values of the current solution at $T(k)$ and δ is a parameter which ranges between 0.1 and 0.5.

Let $i(\in C)$ represent an accepted move and $j(\in C)$ represent any move at temperature T, where $C = 1, 2, ..., L$. Also, let $f(i)$ represent the difference between the

temporary solution and the current solution at i. Then $\sigma_{T(k)}$, the variance of the solution at $T(k)$, can be calculated as

$$\sigma_T^2 = E(X^2)_T - E(X)_T^2, \text{ where } X \text{ is a random move}$$

$$\begin{aligned}\sigma_T^2 &= \Sigma_{i\in S} f(i) P_T\{X = i\} - \Sigma_{i\in S} f^2(i) P_T\{X = i\} \\ &= \Sigma_{i\in S} f(i) \frac{e^{-f(i)/T}}{\Sigma_{i\in S} e^{-f(i)/T}} \\ &= \frac{\Sigma_{i\in S} f(i) e^{-f(i)/T}[1 - f(i)]}{\Sigma_{i\in S} e^{-f(i)/T}} \end{aligned} \tag{4.2}$$

4.3.6 The Stopping Criteria

Determining the freezing point of the annealing procedure is another task. In the proposed algorithm the freezing point can be reached by either of the following two ways. Firstly, if there is no improvement in the best objective function for a number of neighbour generation at any temperature level then the ground state has been reached and there is no point in proceeding with the algorithm.

In the second stopping criterion, the annealing is stopped when the probability of accepting a priority assignment, A_c, is less than some probability, ϵ. I.e., we stop if

$$P_a = e^{-(CP_t - CP_c)/T} < \epsilon \tag{4.3}$$

Note that $CP_t > CP_c$ and the duration of activities are integer. Hence,

$$e^{-(CP_t - CP_c)/T} \leq e^{-1/T}$$

Condition 4.3 is satisfied if $e^{-1/T} < \epsilon$ **or** $T < \frac{1}{ln(1/\epsilon)}$.

After combining these two stopping criteria the annealing procedure stops if either no improvement in the objective function is noticed for a number of moves at any temperature level **or** if the temperature drops below $\frac{1}{ln(1/\epsilon)}$, which ever occurs first.

Chapter 5

The Genetic Algorithm

SUMMARY: This chapter gives a background of Genetic Algorithm. The proposed genetic algorithm for the RCPS problem is explained.

A solution methodology based on genetic search is developed, which involves a new crossover operator, a new mutation operator and a new technique for repairing the chromosomes. The following section gives an overview of the Genetic Algorithms.

5.1 Overview of Genetic Algorithms

Genetic Algorithms are robust search technique that mimic the basic principles of natural selection and biological evolution. Genetic Algorithms (GA) were first articulated by Holland [75]. Since then, a lot of attention has been paid to this approach because of its great potential to address combinatorial optimization problems such as the travelling salesman problem and the job shop scheduling problem (Goldberg [76], Biegel and Davern [77], and Sait and Youssef [72]).

The candidate solution of the optimization problem is represented as a *chromosome*. Each chromosome is made up of units called *genes*. Chromosomes are grouped, judiciously, into sets called *population*. The population at a given stage is referred to as *generation*. The survival of a chromosome is judged by its *fitness value*. The fitness value of each chromosome is obtained by a *fitness function* which evaluates the chromosome with respect to the objective function of the optimization problem under consideration. The fitter chromosome has a higher probability of getting *selected*. These selected chromosomes are called *parents* which represent feasible solutions. Parents are mated (usually, in pairs) to produce new strings of feasible solutions called *offspring/children*.

```
algorithm Simple GA;
begin
 generate initial population and compute the fitness of each
 member;
 while termination criteria is not met do
 begin
  select parents from the current population;
  reproduce new population (offsprings) from these selected
  parents;
  Evaluate the fitness of offsprings;
  Replace current population by new population;
 end;
end;
```

Algorithm 10: Pseudo code for a simple genetic algorithm

The pseudo-code for a simple GA is given in Algorithm 10. There are three main steps in this algorithm which have been subjects of many research work right from the inception of GA. These three steps are:

- Creating new chromosomes by mating current chromosomes via some opera-

tors called genetic operators,

- Evaluating each chromosome in the population using a fitness function, and
- Selection of the fittest parents from the evaluated chromosomes of the popultaion.

A proper selection of the above operations does affect the performance of GA. These three issues are dealt with briefly in the subsections to follow.

5.1.1 Genetic Operators

Reproduction, basically, involves the use of genetic operators on the strings of chromosomes. Each genetic operator takes the chosen chromosomes (parents) in pairs and produce new chromosomes (offspring). The most common genetic operators include crossover and mutation.

The Crossover Operator

The crossover operators, viewed in abstract, are operators that combine subparts of two parent chromosomes to produce new children (offspring) thus allowing new points in the search space to be tested. The operation of crossover starts with two parents independently selected at random from the population on the basis of their fitness. The selection is done in such a way that the better an individual's fitness, the more likely it is to be selected. The crossover operation produces two offsprings. Each offspring contains some genetic characteristics from each of its parents. One simple way for achieving crossover is by partially exchanging selected strings between two random points as shown in Figure 5.1. Such a crossover is

termed as two point crossover. For a list of crossover operators and the recent developments in the design of crosssover operators, refer to Davis [78]. The main purpose of the crossover operator is to perform a distributed search, exploring almost all the solution space. Several crossover operators have been designed by researchers for solving the combinatorial optimization problems.

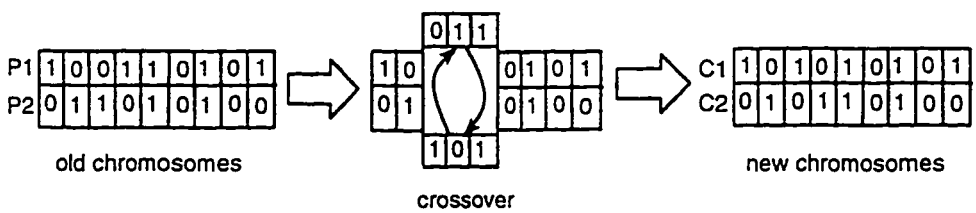

Figure 5.1: Crossover operator in action

The Mutation Operator

Mutation is the secondary search operator which increases the variability of the population. The operation of mutation begins by randomly selecting a chromosome from the population. A mutation point (gene) along the string is chosen at random, and the value assigned to this gene is randomly changed.

The mutation operator offers the opportunity for new genetic material to be introduced into a population. Also, it is potentially useful in restoring genetic diversity that may be lost in a population because of premature convergence. Mutation is used very sparingly in most genetic algorithms work (Koza [79]).

5.1.2 Parent Selection

Parent selection dynamics are based on an application-dependent measure of fitness called fitness function. The purpose of parent selection in a GA is to give more

reproductive chances to those members of the population that are most fit. These selected chrmosomes (called parents) are the fittest of the whole lot and so they live and reproduce children.

One of the most widely used procedures for the above process is the roulette-wheel parent selection algorithm (Goldberg [76]). This algorithm is referred to as roulette-wheel because it can be viewed as allocating pie-shaped slices on a roulette-wheel to population members, with each slice proportional to the member's fitness. To reproduce, we simply spin the wheel, the selected member being the one in whose slice the wheel ends up.

5.1.3 Fitness Function

The fitness function is the link between the genetic algorithm and the optimization problem to be solved. It takes a chromosome as inputs and return a number which is a measure of the chromosome's performance in the problem to be solved. The fitness function plays the same role in genetic algorithms as the environment plays in natural evolution.

Usually, the fitness value is the value of the objective function or some scaled version of it. In general, the fitness function consists of the composition of two functions (Grefenstette and Baker [80]):

$$u(x) = g(f(x)),$$

where f is the objective function and g transforms the value of the objective function to a non-negative number. The fitness function u is often taken as a linear

transformation of h, i.e.,

$$u(x) = af(x) + b,$$

where a is positive if we are maximizing f and negative if we are minimizing, and b is selected to ensure non-negative fitness.

5.2 Genetic Algorithm for the RCPS Problem

In this section, we discuss a GA developed for resource-constrained project scheduling problem. The pseudo-code is given in Algorithm 11. The process of developing GA requires chromosomal representation, initial population, crossover operator, chromosomal repair operator, mutation operator, parent selection and fitness function. These essential components will be discussed separately in the subsections to follow.

```
algorithm RCPS GA;
begin
 set TERM := 0;
 generate initial population of size PS;
 compute the fitness of each chromosome;
 while TERM ≤ NG do
 begin
  select parents from the current population;
  if random number ≤ R_c then apply the crossover operator;
  if random number ≤ R_m then apply the mutation operator;
  Evaluate the fitness of offsprings;
  Replace current population by new population;
 end while;
end;
```

Algorithm 11: Pseudo code for RCPS problem

5.2.1 Chromosomal Representation

In traditional genetic algorithms, chromosomes are represented by binary vectors which are easy to handle. But our problem involves the representation of precedence relations among the project activities, thus making the problem difficult.

For network problems, in general, there are two ways of representations (Davis [78]). One is the indirect representation where chromosomes are coded according to some coding scheme as binary numbers and then converting an ordering of tasks into a legal schedule (Kumar [81]). The second one is the direct representation wherein the schedule itself is used as a chromosome. hence, avoiding the decoding schemes. The latter is also called as the order-based representation where once the order of activities is known then the schedule can be easily determined. The order-based encoding technique has an important advantage over indirect representation in that it rules out a tremendous number of suboptimal solutions (Davis [78]).

For our problem, we used the order-based representation wherein each gene value, p_i, represents the priority of gene (or activity) i. Consider the kth chromosome, C_k, of the current population, then $C_k = \{p_1, p_2, ..., p_N\}$ where N is the number of activities. The values assigned to these genes satisfy the uniformity criterion. I.e., no activity should have a priority less than its predecessors nor greater than its successors. This chromosomal representation will always result in a feasible schedule. Another representation will be to assign the start time of the activity in its corresponding gene. The former representation is used in this paper. I.e., we are using the GA in the priority space/domain and not in the time space/domain.

5.2.2 Initial Population

Generating an initial population is another important step in the design of GAs. Most genetic algorithms work by generating a random initial population. This is not feasible for our problem because of the precedence restrictions or the uniformity criteria. So, a need arises to get the initial population with a blend of randomness and satisfying the uniformity criteira. This is done by using the `Priority` algorithm proposed in Section 4.2.1.

5.2.3 The Crossover Operator

The research done on designing a crossover operator for order-based chromosomes is quite little (Goldberg [76]). Moreover, these operators cannot be applied to our problem because of the precedence constraints. The crossover operator considered here combines the relative orderings of priority of the two parent chromosomes in the two children, similar to the one described by Davis [78] (two point ordered crossover).

Let us suppose that C_x and C_y be the chromosomes selected for the crossover. In other words, C_x and C_y are parents and will reproduce to give two children, C'_x and C'_y. The pseudo-code of the crossover operation is given in Algorithm 12. The crossover operator operates depending on the crossover rate (or probability of crossover) represented as R_c.

As an example, consider the crossover of the following two parents, C_x and C_y, to yield C'_x and C'_y, K_{min} and K_{max} being 3 and 6, respectively as shown in Figure 5.2.

Assign gene values of genes 3,4,5,6 of C_x to genes 3,4,5,6 of C'_x. The remaining

algorithm Crossover;
begin
draw two random integers, U_1 and U_2, between $[2, N-1]$;
set K_{min} := min$\{U_1, U_2\}$ and K_{max} := max$\{U_1, U_2\}$;
set $C'_x := C_x \ \forall\, i \in [K_{min}, K_{max}]$;
permute the remaining gene values of C_x such that they appear in the same;
order as they appear in C_y;
assign these permuted gene values to the empty genes of C'_x;
set $C'_y := C_y \ \forall\, i \notin [K_{min}, K_{max}]$;
permute the remaining gene values of C_y such that they appear in the same;
order as they appear in C_x;
assign these permuted gene values to the empty genes of C'_y;
end crossover;

Algorithm 12: Pseudo code for the crossover operator.

C_x	1	4	6	5	2	3	8	7	9	10
C_y	1	2	3	4	5	8	6	9	7	10
C'_x	1	4	6	5	2	3	8	9	7	10
C'_y	1	2	4	5	3	8	6	9	7	10

Figure 5.2: Example for the proposed crossover operator

gene values (i.e., 1,4,8,7,9,10) are permuted so that they appear in the same order as they appear in C_y. The ordered gene values (i.e., 1,4,8,9,7,10) are then assigned to genes 1,2,7,8,9,10 of C'_x. A similar approach is applied to yield C'_y as shown in Figure 5.2.

5.2.4 The Repair Operator

The proposed crossover operator does not consider the uniformity criteria. Hence, the resulting children are usually discrepant. so, before passing the children back to the population, we repair this discrepancy of the children by applying the repair operator whose pseudo-code is given in Algorithm 13. The uncrossed genes in C'_x and C'_y are those genes which got their gene values directly from C_x and C_y, respectively, and not by permutation. The uncrossed genes, in themselves, satisfy the uniformity criteria. But the positions that these chunks of uncrossed genes occupy on C'_x and C'_y are quite opposite. Hence, repairing of C'_x and C'_y is done in a slightly different fashion.

```
algorithm repair;
begin
 mark all the uncrossed genes, including the first and the last
 genes;
 if i is not marked and all predecessors of i are marked
 (∀ i ∉ [K_min, K_max] for C'_x and ∀ i ∈ [K_min, K_max] for C'_y) then use
 backward_repair function;
 if i is not circled and at least one predecessor of i is circled
 (∀ i ∈ [K_min, K_max] for C'_x and ∀ i ∉ [K_min, K_max] for C'_y)then use
 the backward_repair function;
end repair;
```

Algorithm 13: Pseudo code for the repair operator

```
algorithm backward_repair;
begin
 set index := i and a := p_i;
 while a <priority of immediate predecessor(s) of index do
 begin
  p_MAX := maximum priority among all the immediate predecessors
  of index;
  gene corresponding to p_MAX be MAX;
  switch the position of a and p_MAX;
  index := MAX;
 end while;
 mark gene i and circle it;
end backward_repair;
```

Algorithm 14: Pseudo code for function `backward_repair`

5.2.5 The Mutation Operator

The side effect of using the crossover operator along with parent selection is that we might loose some potentially useful information from the chromosome. To avoid this side effects we introduce mutation operator. The mutation operator used for our problem is similar to the one used in Section 4.2.4. The pseudo-code is given in Algorithm 15. The mutation operator is activated depending on the mutation rate, R_m. This operator mutates the chromosome and also repairs it simultaneously. The mutation is carried out by switching the priorities of two randomly selected parallel genes. Two genes, r and s, are parallel if none of them is a predecessor of the other. For a detail analysis of this operator refer to Section 4.2.4.

5.2.6 Parent Selection and Fitness Function

As mentioned earlier in the previous section that the roulette wheel parent selection mechanism is the most common technique in use. So, we also used the same

```
algorithm mutation;
begin
 select two parallel genes r and s with gene values p_r and
 p_s (p_r < p_s);
 switch the gene values of genes r and s;
 use the backward_repair algorithm with i := s;
 use the forward_repair algorithm with i := r;
end mutation;
```

Algorithm 15: Pseudo-code for the mutation operator

```
algorithm forward_repair;
begin
  set index := i and b := p_i;
  while b > p_index do
  begin
    p_MIN := minimum priority among all the immediate successors
    of index;
    gene corresponding to p_MIN be MIN;
    switch the position of b and p_MIN;
    index := MIN;
  end while;
end forward_repair;
```

Algorithm 16: Pseudo code for function forward_repair

mechanism integrated with the alias method (Kornmal and Peterson [82]).

The project completion time, f, is converted to a fitness function as per the following tranformation:

$$g(f(C_k)) = \frac{f_{max} - f(C_k) + \gamma}{f_{max} - f_{min} + \gamma} \tag{5.1}$$

where $f(C_k)$ is the objective function of kth chromosome of the current generation, f_{max} is the maximum objective function in the current generation and f_{min} is the minimum objective function in the current generation. γ is a scalar which can have a value in the range (0,1) and is basically used to avoid zero division in (5.1).

The fitness value is then calculated as $g(C_k)/\Sigma_{i=1}^{PS} g(C_i)$. The denominator is the sum of all the fitness function of the chromosomes in the current population of size PS. The children with best fitness value are carried over to the next generation by replacing the worst parents of the current generation. This strategy is called as the *elitist* strategy (Goldberg [76]).

As is clear from this section that the following choices must be made for the implementation of the proposed GA:

- Number of generations (NG),
- Population size (PS),
- Probability of mutation (R_m), and
- Probability of crossover (R_c),

Chapter 6

A Tabu Search Procedure

SUMMARY: This chapter gives a background on Tabu Search. The proposed Tabu Search procedure for the RCPS problem is then explained in detail.

6.1 Tabu Search

A general iterative technique, called Tabu Search (TS), was proposed by Glover [84, 85] for finding good solutions to combinatorial optimization problems. This technique is conceptually simple and elegant. Glover [85] defined TS as a meta-heuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality.

Within the basic concept of TS we try to improve a given feasible solution by *transforming* it iteratively into other feasible solutions while allowing for intermediate deteriorations of the objective function. Transformations are referred to as *moves* and may be described by a set of one or more *attributes*. *Tabu restrictions* are those

restrictions that classify certain moves as forbidden, while *aspiration criteria* make the algorithm capable of overriding the tabu status of moves.

Whenever hill climbing search procedure, alone, is applied to a combinatorial optimization problem it might get trapped in a local optimum. The same procedure when applied, in conjunction with TS, to the same optimization problem does not get terminated upon reaching a point of local optimality. This is accomplished by forbidding moves with certain attributes (making them tabu), and choosing moves from the unforbidden ones. In this respect, the method is a constrained search procedure, where each step consists of solving a secondary optimization problem, allowing only those solutions (or moves) that are not forbidden or tabu (Glover [85], Glover and Greenberg [86], Glover et al. [87]).

There are two main underlying objectives of using TS as enumerated below.

- To prevent cycling, and
- To explore new regions.

Obviously, to prevent cycling, we need to store previous moves or assignments. Thus, TS is not a Markovian (or memoryless) search technique and has a flexible memory structure in conjunction with strategic restrictions and aspiration levels as a means for exploiting the historical search space more throughly than by techniques using rigid memory structures (such as branch and bound and A* search) or by memoryless systems (such as simulated annealing).

The success of the method has been noteworthy. Many researchers have made contributions to advance the field in past several years (Glover [85], Glover and Greenberg [86], Glover et al. [87] Sait and Youssef [72]). Some of the diversity of

```
algorithm simple TS;
begin
 generate an initial feasible solution;
 initialize TLS and NITER;
 set i := 1;
 while i ≤ NITER do
 begin
  determine some admissible moves from current solution;
  evaluate the solutions resulting from the admissible moves;
  perform the best admissible move;
  add the best move to tabu list;
  if number of entries in tabu list greater than TLS then delete
  the oldest entry;
  i := i + 1;
 end while;
end simple TS;
```

Algorithm 17: Pseudo-code showing a simple tabu search procedure.

tabu search applications include, but not limited to, job-shop scheduling, assembly-line balancing, cluster analysis, knapsack problem and JIT production. A simple TS procedure is given in Algorithm 17. The procedure starts with an initial feasible solution set and sets the tabu list size (TLS) and the number of iterations ($NITER$). In each iteration, the procedure determines the admissible moves and evaluates the solutions resulting from the admissible moves. The best admissible move (which is not tabu or tabu but satisfy the aspiration criteria) is performed and added to the tabu list. If the number of elements that are in tabu list exceeds TLS then the oldest element in the list will be replaced with a new element. The algorithm will continue till the number of iterations is greater than $NITER$

6.1.1 Attributes

Attributes are generally defined such that each has a natural complement or 'reverse' attributes. A list of such attributes which are recorded in the same sequence as their corresponding solutions are generated is called a tabu list. Broadly speaking, this list defines a set of constriants that must be satisfied by all admissible moves while the elements of the tabu list are in effect.

6.1.2 Short Term Memory Component

Tabu search may be viewed as a nested hierarchy of long-, intermediate- and short-term memory functions, with the short-term function forming the core of the procedure (Glover [91]). The intermediate and long-term memory functions of a tabu search coordinate successive passes of the short-term memory.

The most common used short-term memory keeps track of solution attributes that have changed during the recent past, and is called recency-based memory. Recency-based memory is managed by creating one or more tabu lists, which record the moves made in recent past. The short-term or recency-based memory component operates by selecting moves designed to progress quickly to a local optimum and then go beyond local optimum by forbidding moves with certain attributes. No concern is given to the fact that the best moves available may not improve the current solution. Instead, the method selects the moves with highest evaluations, from the set not in tabu list, to push the search in new regions.

The most difficult part of implementing TS is finding an appropriate tabu list size. Too small tabu list may lead to cycling and too large tabu list may be too restrictive. No rule exists that gives good tabu list size (Glover [91]).

6.1.3 Aspiration Level and Aspiration Criteria

The aspiration levels provide thresholds of attractiveness that govern whether the solutions may be considered admissible inspite of being classified tabu. If a move, x', is made then the new objective function value, $C(x')$, is recorded and the move is added to the tabu list. A reversal of x' (say x'') can occur if $C(x'')$ is better than $C(x')$. This criteria is termed as aspiration criteria. The aspiration level of attribute x' is $C(x')$. Aspiration criteria must make sure that reversal is leading to a solution which is better and is not the same as the previous one otherwise cycling can occur.

6.2 A Tabu Search Procedure for the RCPS Problem

In this section, we discuss a new TS procedure proposed for RCPS problem. In the proposed algorithm each activity will be assigned some priority, the generation of neighbors will be acheived by changing the priority assignment. Essentially, the tabu search is conducted in the *priority space*. The aspiration level will be taken as the project duration resulting from scheduling the activities according to the assigned priority. Our main goal is to find how local search method can be applied when certain moves in the search space are restricted. Our solution procedure consists of storing the most recent moves using a short-term memory structure. The algorithmic description of the proposed TS procedure, called TSRCPSP, is given in Algorithm 18.

In this algorithm we use activity-on-node representation. Moreover, the labels of the nodes are in increasing order. I.e., a node will have a label smaller than its

```
algorithm TSRCPSP;
begin
 generate an initial feasible solution s ∈ X;
 set TL := { }, AL := objective function of s and i := 1 ;
 while i ≤ NITER do
 begin
  set count := 0;
  generate NNS number of neighbour solutions S(s) ∈ N(s);
  select best s' ∈ S(s);
  count := count + 1;
  if move s' to s is not in TL then
  begin
   accept move and update best solution;
   update TL and AL;
   i := i + 1;
  end if;
  else if c(s') < AL then
  begin
   accept move and update best solution;
   update TL and AL;
   i := i + 1;
  end if;
  else if count < NNS then go back and select next best s' ∈ S(s);
  i := i + 1;
 end while;
end TSRCPSP;
```

Algorithm 18: Pseudo-code for the proposed tabu search procedure

successor(s) and greater than its predecessor(s). Furthermore, for each node, a list of all its predecessors is generated. The search starts by generating an initial feasible solution $s \in X$ where X represent a set of feasible solutions. The tabu list (TL) and the aspiration level (AL) are initialized in the next step. Let $N(s)$ be the set of neighboring solutions of x and $S(s)$ be a sample of neighboring solutions of s with sample size equal to NNS. Selecting the best admissible move (s'), where $s' \in S(s)$. is the next step after which a check is made whether the move s' to s is in TL or not.

- If it is not in TL then the move is accepted and TL and AL are updated.
- If it is in TL then either of the following two cases might arise.
  - If the objective function (project duration) of the move s', represented as $c(s')$ is less than AL then the move is accepted and TL and AL are updated.
  - If $c(s') \geq AL$ then the move is rejected and the next best neighbour from $S(s)$ is selected and the procedure is repeated.

The procedure is stopped when the number of iterations reaches a value which is equal to $NITER$. The following subsections deal with the elements of the above algorithm, i.e., generation of initial feasible solution, generation of neighbors, tabu list management and the parameters of the algorithm.

6.2.1 Generation of Initial Feasible Solution

Initially each activity is assigned a unique priority such that a priority of an activity is higher than any of its successors. These priorities guarantees that an activity is

not a candidate for schedule before its predecessors. The duration or critical path of the project is obtained by scheduling the activities according to their respective priority within the available resource constraints.

6.2.2 Generation of Feasible Neighbors

Generation of the feasible neighbors of the current solution (s) is the most important task of tabu search procedures. The procedure that we used in our approach is the one mentioned in Chapter 4, i.e., the algorithm `switching`. The critical path of the resulting prioritized network is then calculated.

6.2.3 Tabu List Management

To avoid revisiting a solution, a check is kept on the reversal of the priorities by maintaining a tabu list in which the switched nodes and their respective priorities (attributes) are entered as shown below.

N_a	a	N_b	b

While maintaining this list we follow first-in-first-out (FIFO) rule. Thus, this list can be viewed as a spider's web as shown in Figure 6.2, where four concentric circles represent N_a, a, N_b, and b in this order, starting from the innermost circle. The circled numbers outside the web contain the aspiration level for that move. The aspiration level of a move is equal to the length of the critical path of the network resulting from switching the priorities of two nodes using algorithm `switching`. A priority switching (move) which is restricted (or tabu) is accepted if it satisfies the aspiration criteria. This means that the reversal of the move is leading to a smaller critical path duration and is readily accepted.

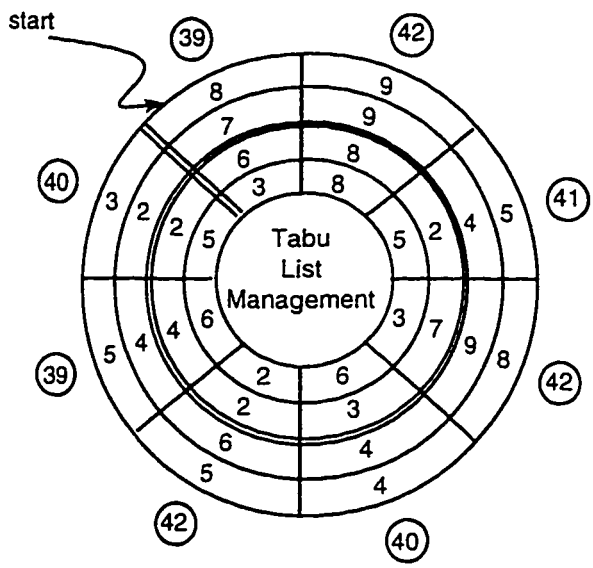

Figure 6.1: Spider's web like tabu list

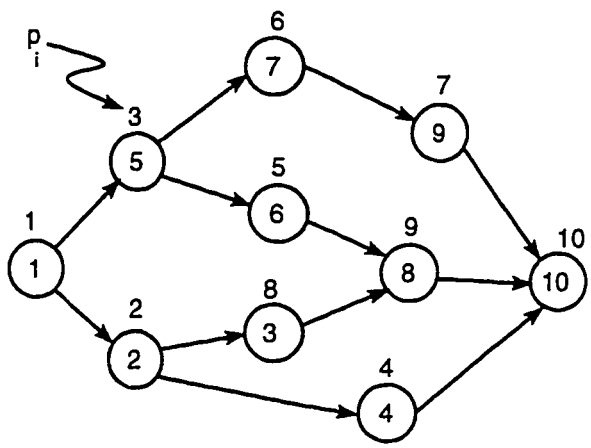

Figure 6.2: Activity network with current assignment of priorities

As an example, Figure 6.2 shows the current tabu list. The network in Figure 6.3 shows the priorities of the activities (nodes). Consider switching the priorities of nodes 3 and 7. Suppose that the corresponding duration of the critical path is calculated as 40. This move is tabu and its aspiration level is 39 as shown in Figure 6.2. The move is rejected. Consider another move when $N_a = 2$ and $N_b = 6$. Suppose that the corresponding duration of the critical path is 41. This move is tabu (or restricted) but the aspiration level on Figure 6.2 is 42. Hence this move is accepted and the aspiration level is changed from 42 to 41.

6.2.4 Parameters of TS Algorithm

In the proposed TS procedure for the RCPS problem, their are three main parameters which effect the performance of the algorithm. These three parameters are tabu list size, number of neighbors to be generated at each iteration, and number of iterations. In the following subsections we discuss these parameters.

Tabu List Size (TLS)

Empirically, tabu list sizes that provide good results often grow with the size of the problem. However, no single rule gives a good size for all classes of problems. Stronger restrictions are generally coupled with smaller sizes. The way to identify a good tabu list size for a given problem class and choice of tabu restrictions is simply to watch for the occurence of cycling when the size is too small and the deterioration in solution quality when the size is too large (Glover [91]). Best sizes lie in an intermediate range between these two extremes.

Number of Neighbors Generated (NNS)

Ideally one would like to perform a comprehensive neighbourhood examination for best solution. This, in general, is very expensive in terms of computation time. Also, there is a possibility of revisiting a solution. A less neighbourhood examination saves computation time but may effect the quality of solutions. So, a good selection of the value of NNS is required for producing quality solutions within a reasonable computation time.

Number of Iterations ($NITER$)

The number of iterations should be as large as possible to move to as many neighbourhoods, in the solution space (priority space for our procedure), as possible. This choice will require long computation time. A low $NITER$ will leave us with only small number of neighbourhoods visited, thus effecting the quality of the solutions. On the other hand, a very large $NITER$ may lead to cycling as the moves get freed from restrictions (tabu).

Chapter 7

Computational Results And Comparisons

Summary: This chapter deals with the computational results of the three proposed algorithms based on simulated annealing, genetic algorithm and Tabu search. A comparison is also made with the popular procedures.

The proposed algorithms described in Chapter 4, 5 and 6 have been coded in C on the UNIX environment. Execution of the programs has been done on the Pentium machines networked with UNIX operating system with Intel 586 processor running at 66 MHz.

7.1 Patterson's 110 Standard Problems

One hundred and ten benchmark problems assembled by Patterson [52] were used to test the algorithms. These problems represent an accumulation of all multi-resource

problems existing in the literature till 1984. Plus, some problems were included in the problem data base that have been optimally solved for the first time. The number of activities included in these test problems varies between 7 and 51, with the number of resource types required per activity varying between one and three. The majority of the projects (103) consist of activities which require the use of the full complement of three different resource types for their performance.

This problem data set can be categorized as shown in Table 7.1. This data set is being used by many researchers of RCPS (Ozdamar & Ulusoy [2]). The best solution obtained for all these problems are also given in the literature.

Table 7.1: Classification of Patterson 110 problem data set

Number of Activities	Number of types of resources	Number of Problems
7	3	1
8	2	2
9	1	2
13	3	1
14	3	1
18	1	1
22	3	46
23	3	1
27	3	43
35	1	1
35	2	1
51	3	10

For comparison, the performance measure used is the Sum of Percent Deviation

(SPD) which is given as

$$SPD = 100\Sigma_{i=1}^{110}\frac{O_i - O_i^1}{O_i^1} \tag{7.1}$$

where O_i is the project duration of the ith sample problem obtained using the proposed procedure and O_i^1 is the known optimal project duration for the ith sample problem.

7.2 Performance of Simulated Annealing Algorithm

For obtaining the parameters of the SA algorithm we designed four experiments. In the first experiment the initial chain length, L_o, was kept constant at 700 and the initial temperature, $T(0)$, was varied from 25 to 300 in steps of 25 when $\delta_a = 90$, $\Delta = 50$ and $\epsilon = 0.01$. The results are tabulated in Table 7.2. The minimum SPD occurs at $T(0) = 125$ which remains constant with $T(0) > 125$.

In the second experiment, $T(0)$ was fixed at 125 with $\delta_a = 90$, $\Delta = 50$ and $\epsilon = 0.01$. The chain length, L_o, was varied from 100 to 700 in steps of 100. The results are tabulated in Table 7.3. A general trend of linear decrease in SPD with increase in L_o was observed as anticpated by Lundy and Mees [92]).

In the third experiment, δ_a was varied as 75%, 80%, 85%, 90%, 95% and 99%, keeping all the other values constant, i.e., $L_o = 400$, Δ=50, $T(0) = 125$, $\epsilon = 0.01$. The observations are given in Table 7.4. The best SPD is for $\delta_a = 90\%$.

Finally, in the fourth experiment the value of ϵ was varied as 0.050, 0.025, 0.010,

Table 7.2: Results of experiment 1 on the Patterson 110 problems with $L_o = 700$ $\delta_a = 90$, $\Delta = 50$ and $\epsilon = 0.01$

$T(0)$	Sum of % Deviation
25	101.36
50	82.23
75	69.20
100	64.56
125	58.67
150	58.67
175	58.67
200	58.67
225	58.67
250	58.67
275	58.67
300	58.67

Table 7.3: Results of experiment 2 on the Patterson 110 problems with $T(0) = 125$ $\delta_a = 90$, $\Delta = 50$ and $\epsilon = 0.01$

L_o	Sum of % Deviation
100	65.88
200	59.26
300	58.99
400	58.67
500	58.67
600	58.67
700	58.67

Table 7.4: Results of experiment 3 on the Patterson 110 problems with $T(0) = 125$ $L_o = 400$, $\Delta = 50$ and $\epsilon = 0.01$

δ_a	Sum of % Deviation
75	63.01
80	61.98
85	60.10
90	58.67
95	58.67
99	58.67

Table 7.5: Results of experiment 4 on the Patterson 110 problems with $T(0) = 125$ $L_o = 400$, $\Delta = 50$ and $\delta_a = 90$

ϵ	Sum of % Deviation
0.050	69.09
0.025	61.88
0.010	58.67
0.005	58.67

and 0.005, keeping all the other values constant, i.e., $L_o = 400$, $\delta_a = 90$, Δ=50, $T(0) = 125$. The observations are given in Table 7.5. With the increase in value of ϵ the corresponding freezing temperature increases effecting the quality of the solution, i.e., the annealing procedure is being stopped much before the ground state is reached and hence the bad quality with high values of ϵ.

As far as the value of δ_a is concerned, Figure 7.1 confirm the remarks made earlier that at high temperature we accept almost all the moves to explore the priority domain. But as the number of chains increases the temperature is reduced slowly (analogous to annealing of solids) and the acceptance percentage also decreases to zero at low temperature levels. The plot of the temperature decrement function is also shown in Figure 7.2.

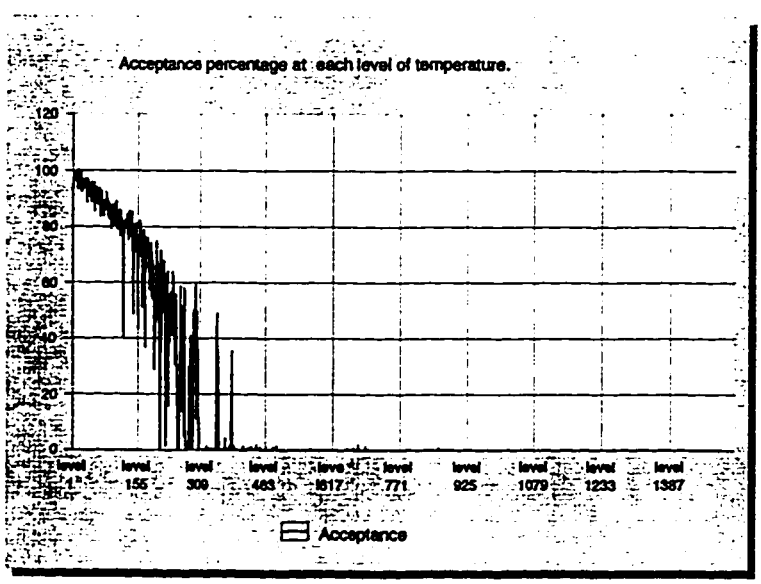

Figure 7.1: Acceptance Percentage Decrement

For testing the SA algorithm by making the initial temperature as a function of problem size, the experiment, as shown in Table 7.6, is devised. As can be seen from this table that for the projects which have less than 20 the best $T(0)$ is 40. For the

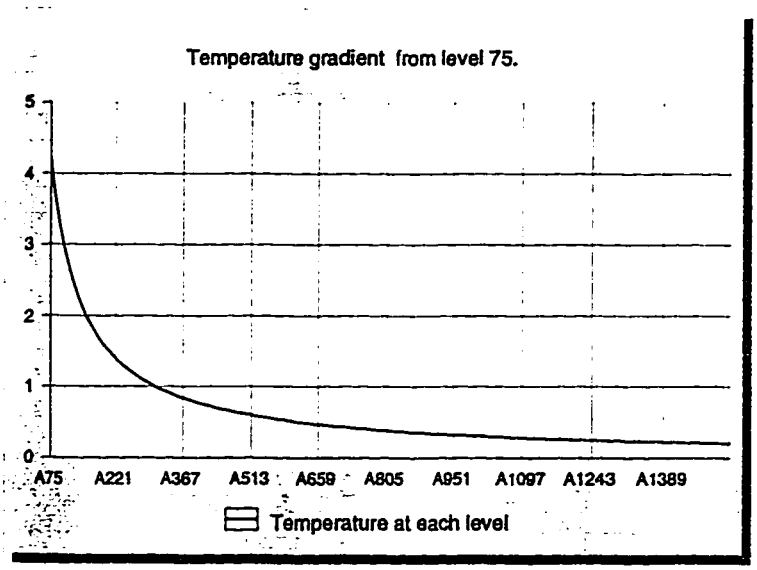

Figure 7.2: Temperature Function

Table 7.6: Effect of varying initial temperature on SPD

Problem Class	$T(0)$	SPD
less than 20 activities	20	17.33
	30	16.57
	40	10.52
	50	10.52
	60	10.52
between 21 to 25 activities	20	40.63
	40	36.79
	60	25.32
	80	17.00
	100	17.00
between 27 to 51 activities	40	108.37
	60	56.39
	80	48.66
	100	37.65
	120	31.15
	140	31.15

network with activities ranging from 21 to 25 the best $T(0)$ is 80 and for those in the range 27 to 51 the best $T(0)$ is 120.

7.3 Performance of Genetic Algorithm

The proposed genetic algorithm was applied to the 110 problem set of Patterson [52]. The most difficult and time-intensive issue in the successful implementation of GAs is to find good parameter settings. Davis [78] enumerated a number of approaches including a brute-force search, using GAs at meta level, and using adaptive operator fitness technique. We used **response surface methodology (RSM)**, an effective local search technique used in the statistical design and analysis of experiments.

Response surface methodology is essentially a particular set of mathematical and statistical methods used to aid in the solution of certain types of problems where a number of (controllable) variables in some system influence the response of the system and the goal is to optimize this response. The response is normally measured on a continuous scale and is a variable which likely represents the most important function of the system. Generally, we do not know the mathematical relation between the controllable variables and the response. If we have k controllable variables then the surface represented by $\eta = \eta(x_1, x_2, ..., x_k)$ is called a response surface. Two things are to be noted here:

- the form of η is unknown and perhaps extremely complicated, and
- the values of $x_1, x_2, x_3, ..., x_k$ which optimizes η are also unknown.

For a good overview on RSM refer to Montgomery [93] and Myers et al. [94].

Parameters or factors	Notation
Number of generations	NG
Population size	PS
Mutation rate	R_m
Crossover rate	R_c

Table 7.7: Variables for the response surface design

In our case the response is the sum of percent deviation (SPD) of all 110 problems and the controllable variables are as shown in Table 7.7. For deciding about the design points (i.e., levels of each factor) the suggestion given by Montgomery [93] is to take extreme values. Since this decision of choosing the levels of the factors/parameters is important, we decided to do some preliminary runs before getting into response surface methodology.

The selection of the value of γ was the first step before making the preliminary runs. In the literature it is mentioned that the value of γ should be between 0 and 1. It is recommended that $\gamma = 0.1$ yields better result (Davis [78]).

For performing a preliminary study about the search space we designed several 2^k factorial designs with different levels for each factor. Since we had $k = 4$ factors where each of the k factors is at two levels so we need to have $2^4 = 16$ runs with one replication at each design point. The minimum response, i.e. 121.8, that we got in all of these designs were for $NG = 100$, $PS = 150$, $R_m = 0.8$ and $R_c = 0.9$.

So, we decided to anchor on this point as the center point and narrow down our search on the space just around this point. By inspecting the preliminary design points we decided upon the levels of each factor to be as shown in Table 7.8. The response at each factor level combinations are given in Table 7.9 and the fitted

Parameters or factors	Levels	
	Low	High
Number of generations (NG)	90	110
Population size (PS)	140	160
Mutation rate (R_m)	0.7	0.9
Crossover rate (R_c)	0.85	0.95

Table 7.8: Levels of final design

Factor				Response
NG	PS	R_m	R_c	SPD
90	140	0.7	0.85	147.38
90	140	0.7	0.95	131.74*
90	140	0.9	0.85	156.69
90	140	0.9	0.95	154.20
90	160	0.7	0.85	144.19
90	160	0.7	0.95	142.13
90	160	0.9	0.85	148.82
90	160	0.9	0.95	148.00
110	140	0.7	0.85	144.05
110	140	0.7	0.95	131.74*
110	140	0.9	0.85	156.69
110	140	0.9	0.95	154.20
110	160	0.7	0.85	144.19
110	160	0.7	0.95	142.13
110	160	0.9	0.85	147.56
110	160	0.9	0.95	148.00

Table 7.9: Response for each factor level combination

Independent variable	Coefficient	Std. error	t_o	Sig. level
constant	151.605	32.1028	4.7225	0.0006
NG	-0.02869	0.120142	-0.2388	0.8157
PS	-0.134312	0.120142	-1.1180	0.2874
R_m	60.26875	12.014162	5.0165	0.0004
R_c	-34.5125	24.028325	-1.4363	0.1787

Table 7.10: Linear multi-factor regression model

Source	SS	df	MS	$F_o(\alpha = .05)$
regression	658.997	4	164.749	7.13372
error	254.039	11	23.0944	
total	913.035	15	$R^2 = .721765$	
Main Effects:				
NG	1.31676	1	1.31676	0.118
PS	28.86376	1	28.86376	2.581
R_m	581.17156	1	581.17156	51.971
R_c	47.64451	1	47.64451	4.261

Table 7.11: ANOVA for the final design levels

Steps	NG	PS	R_m	R_c	SPD
Δ	0	0	-0.02	0.01	
origin	100	150	0.8	0.9	121.8
origin+Δ	100	150	0.78	0.91	97.76
origin+2Δ	100	150	0.76	0.92	95.75*
origin+3Δ	100	150	0.74	0.93	119.67

Table 7.12: Steepest Ascent experiment

regression model is given in Table 7.10. The model adequacy check was performed by the residual analysis and it revealed that there is no major difficulty with the fitted model. So, we conclude that the multiple linear regression model is an adequate fit. Also, the test statistic F_o is greater than $F_{\alpha.k.n-k-1}$ and hence we say that atleast one variable contributes significantly to the regression. From Table 7.11 it is clear that factor R_m (i.e., mutation rate) is that variable which contributes significantly to the regression. Hence, for performing the steepest descent experiment a step size (Δ) of 0.02 for factor R_m is taken and the step sizes for the other factors are calculated which is proportional to their regression coefficients. The observations are recorded in Table 7.12. The minimum response of 95.75 is got at (100, 150, 0.76, 0.92). Since the step size (Δ) for factor NG and factor PS was found to be zero we were eager to see the variation in response if the levels of factor NG and factor PS are varied in the range ± 5 from the center point (i.e., moving in a ridge). The observations are given in Table 7.13. Thus, we conclude that there is no significant improvement in the minimum response in the ridge. Hence, the good (but not the best) parameter setting found so far is (100, 150, 0.76, 0.92).

NG	PS	R_m	R_c	SPD
95	145	0.76	0.92	109.84
95	155	0.76	0.92	113.33
105	145	0.76	0.92	97.72
105	155	0.76	0.92	107.37

Table 7.13: Moving in a ridge

7.4 Performance of Tabu Search procedure

The proposed tabu search procedure when implemented on Patterson's 110 problem data set gave the best results obtained so far. The successful implementation of the TS procedure can be accomplished by good parameter choice. In this section we will design an experiment to study the significance of the parameter values.

We examined the effects of variations in parameters, $TLS, NITER$ and NNS, and the various interactions among them on the average critical path length of all the 110 problem of Patterson. The statistical analysis were performed using analysis of variance (ANOVA).

After an extensive preliminary anlysis, we chose three levels for each parameter which represented the low, medium and high levels of these paramters (hereafter will be called as factors), as shown in Table 7.14. Thus, we have 3^k factorial design with single replication. The observations are tabulated in Table 7.15. The multiple regression model and the ANOVA done on these observations are summarized in Table 7.16 and 7.17, respectively, with $\alpha = 0.05$. The three-way interaction is taken as error estimator. An examination of the F-statistic for regression reveals that $F_o > F_{\alpha,k,n-k-1}$ and hence at least one factor is significant. Observing the main effects, one can say that all the main effects have high impact on the solution value.

Table 7.14: Levels of 3^k design for tabu search parameters

Parameters or factors	Levels		
	Low	Medium	High
TLS	3	7	50
NITER	50	100	125
NNS	100	150	200

In other words, all the main effects are significant at $\alpha = 0.05$. The TLS and $NITER$ interaction effect is also quite significant while the other two interactions are not. In general, the following guidelines are given for the selection of best parameters. The tabu list size should be moderate enough just to avoid revisiting a solution. If the tabu list size is too high then it becomes too restrictive and might effect the quality of the solution. The "magic seven" value of the list size as mentioned by Glover et al. [86] performs quite well for our procedure as well. As deduced from the statistical analysis, the number of iterations should be high, 125. The number of neighbouring solutions explored in each iteration should be moderate, 100.

An analysis was also done to study the effect of the size of the problem on the optimal parameter. Table 7.18 shows the results. For problems of size less than 20 activities the best parameter setting was found to be $TLS = 3$, $NITER = 30$, and $NNS = 50$. For large problems $TS = 7$, $NITER = 125$, and $NNS = 100$.

Table 7.15: Observations made on each combination of factor levels

Factors			Solution
TLS	*NITER*	*NNS*	
3	50	100	89.08
3	50	150	96.44
3	50	200	86.71
3	100	100	72.44
3	100	150	87.76
3	100	200	82.30
3	125	100	66.21
3	125	150	72.46
3	125	200	65.76
7	50	100	86.11
7	50	150	96.44
7	50	200	84.93
7	100	100	70.52
7	100	150	78.83
7	100	200	74.21
7	125	100	62.64*
7	125	150	72.46
7	125	200	65.76
50	50	100	97.57
50	50	150	92.87
50	50	200	90.01
50	100	100	75.50
50	100	150	85.70
50	100	200	74.21
50	125	100	72.73
50	125	150	72.46
50	125	200	65.76

Table 7.16: Linear multi-factor regression model for TS parameters

Independent variable	Coefficient	Std. error	t_o	Sig. level
constant	105.9406	4.8378	21.8983	0.0000
A	0.0478	0.0464	1.0298	0.3138
B	-0.2966	0.03165	-9.3710	0.0000
C	-0.0035	0.02417	-0.1448	0.8862

Table 7.17: ANOVA for the TS parameters

Source	SS	df	MS	$F_o(\alpha = .05)$	sig. level
regression	2338.09	3	779.36	29.6323	0.0000
error	604.93	23	26.30		
total	2943.02	26			
Main Effects:					
A	74.8273	2	37.4137	4.971	0.0395
B	2330.92	2	1165.4600	154.842	0.0000
C	305.8104	2	152.9052	20.315	0.0007
Interactions:					
AB	32.3759	4	8.0940	1.075	0.4291
AC	77.4434	4	19.3608	2.572	0.1190
BC	61.4300	4	15.3515	2.040	0.1813
Error	60.2142	8	7.5268		

Table 7.18: Effect of TS parameters on the size of the problems

Number of activities	TLS	$NITER$	NNS	Solution
≤ 20	3	25	25	17.33
	3	25	50	18.37
	3	30	25	15.36
	3	30	50	10.52
	5	25	25	17.33
	5	25	50	18.37
	5	30	25	10.52
	5	30	50	10.52
≥ 21	3	100	100	61.92
	3	100	150	77.24
	3	125	100	55.69
	3	125	150	61.94
	7	100	100	60.52
	7	100	150	68.31
	7	125	100	52.12
	7	125	150	61.94

7.5 Comparison of the Proposed GA with Cheng and Gen's GA

Cheng and Gen [62, 63] proposed a genetic algorithm for the resource-constrained project scheduling problem. They reported only the optimum values they got for two test problems consisting of 27 activities from the well known Patterson's 110 data set and compared these optimum solutions with **eight** known heuristics. These eight known heuristics that they compared their algorithms with are:

- Minimum Late Finish Time (LFT),
- Greatest Resource Utilization (GRU),

- Shortest Imminent Operation (SIO),
- Minimum Job Slack (MINSLK),
- Resource Scheduling Method (RSM),
- Select Job Randomly (RAN),
- Most jobs Possible (MJP), and
- Greatest Resource Demand (GRD).

Their solutions for the two problems was equal to the optimum listed in the literature. Their proposed approach outperformed all the eight heuristics. We also got the same optimum solutions for these two test problems. But since they did not gave the CPU time or some other relevant information we cannot compare our results of the 110 problems with their approach.

7.6 Comparison of SA, GA and TS Algorithms with the Popular Procedures

A comparison of the proposed algorithms with the best procedures existing in the literature is done using the Patterson's [52] 110 standard data set. The results are summarized in Table 7.19. It is clear that the proposed algorithms i.e., genetic algorithm, simulated annealing algorithm and tabu search based algorithm performed very well when compared with the best heuristic procedures existing in the literature.

Procedure/Algorithm	% Mean Deviation	Standard Deviation	% of optimal solutions found
Bell & Han's two phase approach	2.60	3.10	44.54
Sampson & Weiss' Local search approach	1.98	2.62	55.45
MINSLACK (initial forward iteration)	5.53	5.66	25.45
MINSLACK (iterative algorithm)	4.66	4.83	27.27
LFT (initial forward iteration)	6.41	5.89	19.09
LFT (iterative algorithm)	5.59	5.38	19.09
WRUP (initial forward iteration)	6.32	5.40	20.00
WRUP (iterative algorithm)	3.77	4.00	31.82
LCBA (initial forward iteration)	3.21	4.41	42.73
LCBA (iterative algorithm)	1.14	2.58	63.63
LI-WI (initial forward iteration)	9.93	7.49	10.90
LI-WI (iterative algorithm)	7.51	6.81	21.82
Proposed GA	0.87	2.035	91.82
Proposed SA algorithm	0.61	1.836	96.36
Proposed TS procedure	0.57	1.740	97.27

Table 7.19: Comparison with the best procedures existing in the literature

Algorithm	Mean Deviation	Standard Deviation	% of optimal solutions found	Mean CPU Time (sec)
Proposed GA	0.87	2.035	91.82	182.73
Proposed SA algorithm	0.61	1.836	96.36	516.66
Proposed TS procedure	0.57	1.740	97.27	75.63

Table 7.20: Comparison of the proposed GA, SA and TS algorithms

7.7 Comparison of SA, GA, TS Algorithms

Finally, we compared the proposed algorithms with each other and the results are tabulated in Table 7.20. As is clear that the Tabu Search based algorithm defeated the genetic algorithm and the simulated annealing algorithm on all the grounds. When the proposed genetic algorithm and the proposed annealing algorithm are compared then a conclusion can be drawn that on the grounds of quality of the solution, the simulated annealing algorithm outperformed the genetic algorithm. But on the grounds of CPU time (sec) the genetic algorithm is far better than the annealing algorithm.

7.8 Justification for the CPU Time

The existing heuristics are very fast but the quality of the solution, usually, is not good. Real life projects are very large with each activity requiring days/months/years for completion. These projects are scheduled much before the start of the project itself (off-line). The losses that will be incurred because of a bad (not optimal) schedule is sometimes very high. Getting a schedule of activities requiring days or months or years for its completion in micro seconds (using the existing heuristics), at the cost of the quality of the solution, is therefore ridiculous. The proposed approaches based on SA, GA, and TS gives a good quality solution in few extra minutes which is very much affordable.

Chapter 8

Contributions and Recommendations

SUMMARY: This chapter deals with the difficulty analysis of the activity networks, in general, and Patterson's 110 data set, in particular. Also, contribution done in this thesis will be highlighted and the scopes for future research will be enumerated.

8.1 Problem Difficulty Analysis

In the literature, resrearchers test their proposed algorithms by using sample problems of particular sizes. For the activity networks, usually, a problem size is specified by the number of activities which constitute a project and nothing is mentioned regarding the precedence relationship may make a project very complex. Actually, these precedence relations make the assignments of the priorities a big issue. If the precedence relation is "simple" then the assignments of the priorities to each node becomes easy, and vice-versa. In this section we analyse this issue in detail.

8.1.1 Classification of Activity Networks

In our approach of finding the difficulty of activity network problem we first classify the activity networks as serial network, parallel network, combination of serial and parallel networks, and compound network. In this subsection we will be dealing with these in detail.

Serial Network

Consider the A-on-N network given in Figure 8.1 (where nodes represents the activities and arcs represents the precedence relations) which represent a serial activity network of 22 nodes and 21 arcs. The completion time of such a project is simply the sum of the durations of all the activities. Also, the priority assignment, with respect to the uniformity criteria, can be done in one and only one way as shown. So, using any method to solve such a resource constrained network is quite ridiculous.

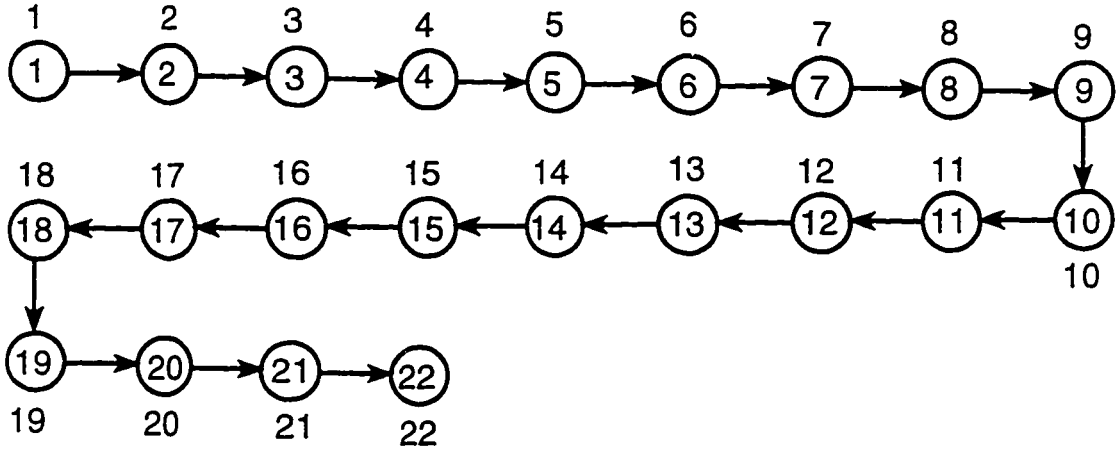

Figure 8.1: Serial network

Parallel network

Consider the A-on-N network given in Figure 8.2 which represents a parallel activity network of 22 nodes and 40 arcs with all the activities in parallel. The priority assignment of such a network can be done in 20! ways. In general, if we have m parallel nodes then the assignment of priorities can be done in $m!$ ways.

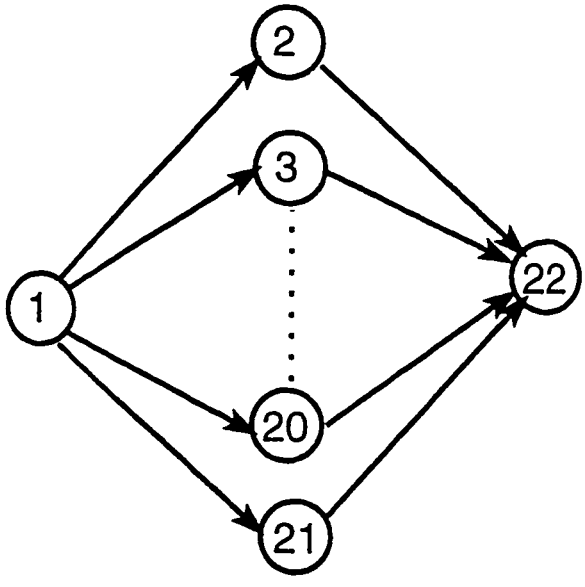

Figure 8.2: Parallel network

Combination of Parallel and Serial Networks

Consider another 22 nodes A-on-N network with 22 arcs shown in Figure 8.3. If the priorities are assigned to one path, the priority assignment for the second path will be fixed and also the source node and sink node always has the first and the last priorities, resectively. In short, the number of possible ways in which priority assignments can be made, considering the uniformity criteria, are $\begin{pmatrix} 20 \\ 10 \end{pmatrix}$.

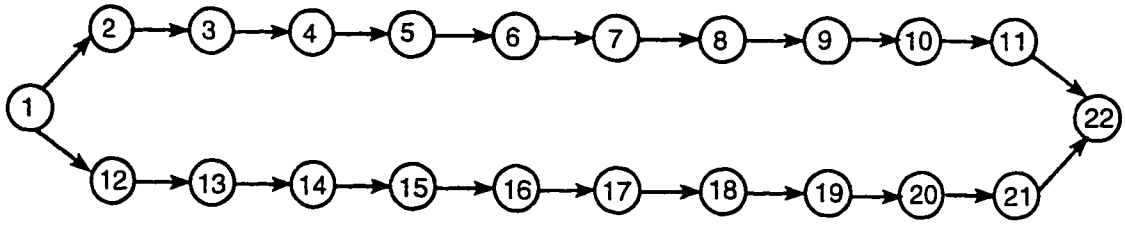

Figure 8.3: Combination of serial and parallel networks

If there are three parallel paths for a 22 nodes network, out of which two paths have seven serial nodes each and the third path have six serial nodes, then the number of possible ways in which priority assignments can be made are

$$\begin{pmatrix} & 20 & \\ 7 & 7 & 6 \end{pmatrix} = \frac{20!}{7!\ 7!\ 6!}$$

In general, if there are $N-2$ nodes (excluding the source and the sink node) in an activity network of k paths with m_i number of serial nodes in path i, where $i = 1, 2, ..., k$, then the number of possible priority assignments is represented as their difficulty factor which is given as

$$\begin{pmatrix} & N-2 & \\ m_1 & m_2 \quad ... & m_k \end{pmatrix} = \frac{(N-2)!}{m_1!\ m_2!\ ...\ m_k!} \text{ where } \Sigma_{i=1}^{k} m_i = N-2 \quad (8.1)$$

Compound Networks

In reality, the activity networks are usually not in the formats as discussed above. They have a complex precedence relationship, as shown in Figure 8.4, which cannot be put into any of the classes mentioned above. Such types of networks are termed

as compound networks. Analysis of compound networks, i.e., finding the number of ways in which the priorities can be assigned, is quite difficult. To make the task of analysing the compound networks a bit easier we propose a procedure which reduces a compound network into a combination of serial and parallel nodes network.

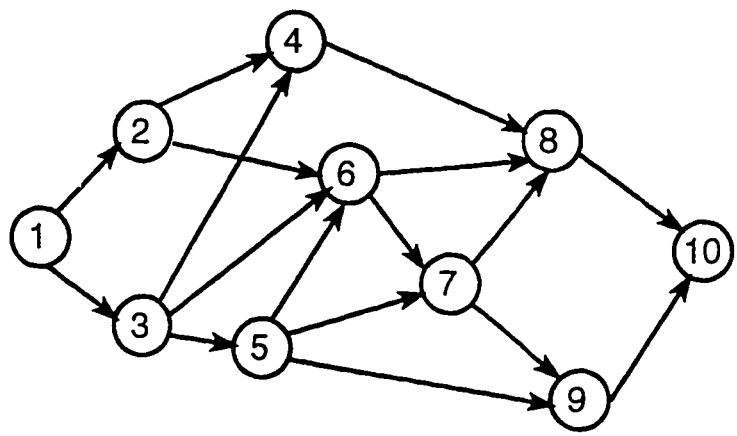

Figure 8.4: Compound network

8.1.2 Difficulty Analysis of Compound Networks: An Upper Bound

A compound network can easily be reduced into a network of serial and parallel nodes by disregarding few arcs in the A-on-N networks. Our main purpose is to find the number of ways of assigning priorities to the nodes of the network. In other words, checking the parallelism of the compound network by developing an upper bound to the number of ways of assigning priorities to it.

For achieving this goal, we assign a processing time of **one** to all the nodes. The critical path (without the resource-constraints) gives the path on which maximum number of nodes are encountered. These critical nodes (except the source and the

sink nodes) are then separated out from the main network and form a serial network. The main network is updated, and on this new network the procedure is repeated untill no node is left (except the source and sink nodes). Finally, the first nodes of all the serial networks, formed from the main network, become the successor of the source node and the final nodes of all these serial networks become the predecessors of the sink node. This way we reduce a compound network into a combination of serial and parallel networks. The compound network will be atleast as tight as the reduced (combination of serial and parallel) network. Thus the number of ways of assigning priorities (difficulty factor) to the nodes of the reduced network is an upper bound for the compound network and is calculated as given by Equation 8.1.

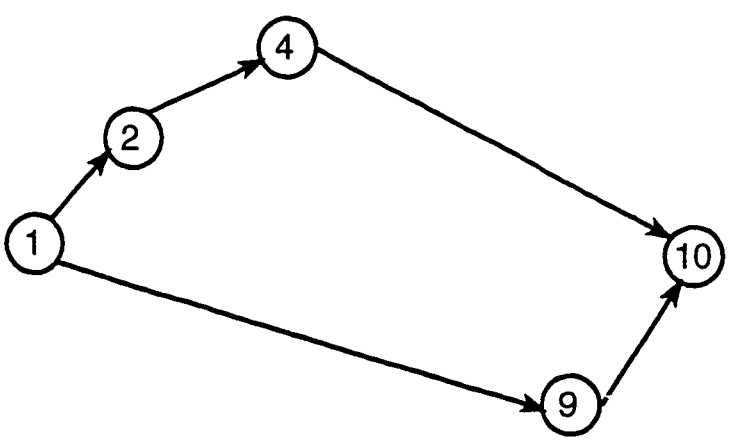

Figure 8.5: The network resulting after removing the critical nodes from the main network

As an illustration, consider the compound network given in Figure 8.4. The critical path of this network is 1-3-5-6-7-8-10 (the path with maximum number of nodes). In case of more than two such paths are available then we select one randomly. The critical activities (nodes 3, 5, 6, 7 and 8) are then removed from the main network and the resulting network is given in Figure 8.5. The critical path

of this network is 1-2-4-10. After removing the critical nodes (2 and 4) we get the network as shown in the Figure 8.6.

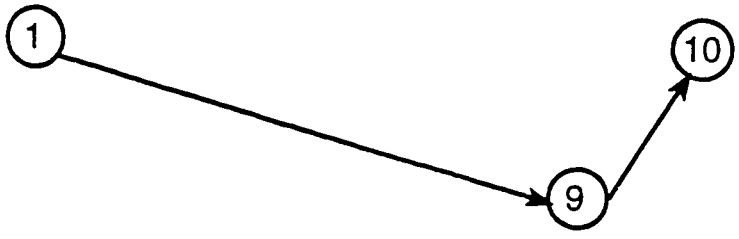

Figure 8.6: The network resulting after removing the critical nodes from the first resulting network

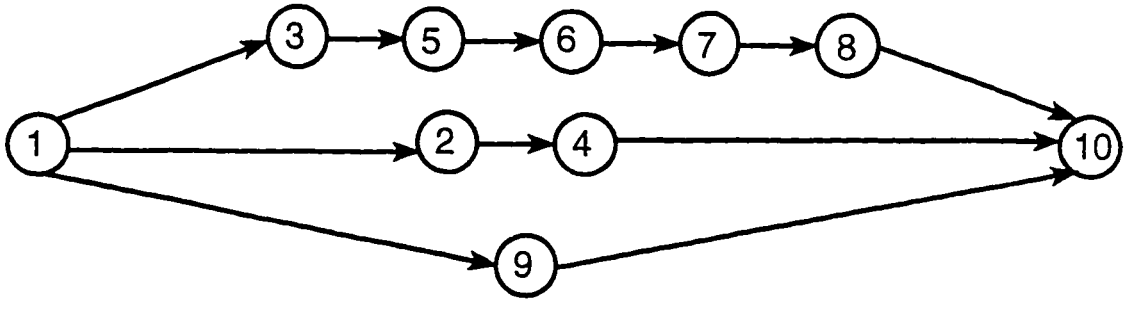

Figure 8.7: The reduced network

The reduced network is given in Figure 8.7 and the number of ways of assigning the priorities are calculated as $\begin{pmatrix} & 8 & \\ 5 & 2 & 1 \end{pmatrix} = \frac{8!}{5!\ 2!\ 1!}$

8.1.3 Difficulty Analysis of Patterson's 110 Problem Data Set

The difficulty analysis of Patterson's 110 problem data set is performed and the results are tabulated in the Table 8.1. The analysis reveals that the problems can be categorised in groups according to their difficulty factor. The problems with

low difficulty factor should be avoided in the implementation of the global search algorithms like GA, SA, and TS since one can enumerate all possible priority assignments.

8.2 Contributions

The following contributions were made in this work.

1. Developed and implemented three new algorithms based on simulated annealing, genetic algorithm and tabu search.

2. Conducted a parametric study of the developed algorithms.

3. The performance of the proposed algorithms were compared with the best heuristic procedures existing in the literature.

4. The difficulty analysis was done for the activity networks, in general, and the Patterson's data set, in particular.

8.3 Recommendations

- Extension of our approach to the multi-project multi-resource project scheduling problem with an objective of minimizing the average completion time of the projects, can be made.

- Extensions can also be made to the Assembly-line balancing problem and the job shop scheduling problem.

Problem no.	no. of Nodes	no. of Arcs	Difficulty of Networks
1	14	20	4.99E+05
2	7	8	20
3	13	15	2.77E+04
4	22	35	3.91E+11
5	22	35	3.91E+11
6	22	35	3.91E+11
7	9	11	420
8	9	11	420
9	18	30	1.21E+09
10	8	11	15
11	8	11	15
12	23	31	2.31E+14
13	22	33	3.52E+14
14	35	55	1.64E+21
15	35	61	2.89E+25
16	22	34	7.82E+11
17	22	35	9.78E+10
18	22	34	7.82E+11
19	22	35	9.78E+10
20	22	32	1.96E+10
21	22	32	1.96E+10
22	22	34	3.91E+11
23	22	33	1.96E+10
24	22	35	3.91E+10
25	22	35	3.91E+10
26	22	35	3.91E+10
27	22	34	4.89E+11
28	22	34	4.89E+11
29	22	35	1.17E+12
30	22	34	1.96E+11
31	22	34	1.96E+11
32	22	32	1.96E+11
33	22	32	1.96E+11
34	22	32	1.96E+11
35	22	32	5.87E+11
36	22	33	3.91E+10
37	22	35	9.78E+10
38	22	35	9.78E+10

Table 8.1: Difficulty analysis of the Patterson 110 problem

Problem no.	no. of Nodes	no. of Arcs	Complexity of Networks
39	22	33	9.78E+10
40	22	34	9.78E+10
41	22	34	1.96E+10
42	22	36	3.91E+11
43	22	36	3.91E+11
44	22	34	2.93E+11
45	22	34	2.93E+11
46	22	33	1.96E+11
47	22	33	1.96E+11
48	22	24	6.52E+09
49	22	24	6.52E+09
50	22	32	6.52E+09
51	22	32	6.52E+09
52	22	40	6.52E+09
53	22	64	6.52E+09
54	22	68	6.52E+09
55	22	68	6.52E+09
56	22	68	6.52E+09
57	22	32	6.52E+09
58	27	43	1.06E+15
59	27	40	4.24E+14
60	27	43	6.36E+15
61	27	42	1.27E+16
62	27	40	3.53E+13
63	27	42	7.63E+16
64	27	40	3.18E+15
65	27	41	2.12E+14
66	27	43	8.48E+15
67	27	42	4.24E+14
68	27	40	6.36E+15
69	27	40	4.24E+14
70	27	39	1.49E+15
71	27	41	8.48E+14
72	27	39	1.48E+15
73	27	41	8.48E+14
74	27	40	2.54E+15
75	27	40	2.54E+15
76	27	40	2.54E+15

Table 8.2: (cond.) Difficulty analysis of the Patterson 110 problem

Problem no.	no. of Nodes	no. of Arcs	Complexity of Networks
77	27	41	8.48E+14
78	27	41	8.48E+14
79	27	41	8.48E+14
80	27	41	8.48E+14
81	27	41	8.48E+14
82	27	41	8.48E+14
83	27	41	8.48E+14
84	27	41	8.48E+14
85	27	41	8.48E+14
86	27	41	8.48E+14
87	27	42	2.54E+15
88	27	44	4.24E+15
89	27	40	8.48E+15
90	27	37	8.48E+14
91	27	38	4.24E+14
92	27	36	7.63E+16
93	27	36	4.24E+13
94	27	38	4.24E+14
95	27	39	3.82E+16
96	27	36	1.27E+16
97	27	35	1.27E+15
98	27	38	4.95E+15
99	27	39	2.67E+17
100	27	37	1.48E+15
101	51	85	7.72E+34
102	51	81	2.94E+36
103	51	87	3.09E+33
104	51	81	3.09E+35
105	51	87	1.95E+39
106	51	78	1.80E+35
107	51	78	2.25E+34
108	51	76	5.40E+35
109	51	80	3.60E+35
110	51	70	5.40E+35

Table 8.3: (cond.) Difficulty analysis of the Patterson 110 problem

- A simulation study can also be made for those project networks where the resources themselves are variable.
- Evolutionary algorithms can be applied for the resource-constrained project scheduling problems.
- A multi-objective approach to minimize the completion time of the project and maximize the net present value (NPV) can be made.

Bibliography

[1] Willis, R. J. Critical path analysis and resource-constrained project scheduling - theory and practice. *European Journal of Operations Research*, 21, 1985.

[2] Ozdamar, L. and Ulusoy, G. A survey on the resource-constrained project scheduling problem. *IIE Trans.*, 27:574–586, 1995.

[3] French, S. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop.* Ellis Horwood ltd., London, 1990.

[4] Blazewicz, J., Lenstra, J. K. and Rinnooy Kaw, A. H. G. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[5] Hall, H. M., and Evans, G. W. A simple method for estimating the duration of a resource-constrained project. *Computers and Industrial Engg.*, 12(1):47–55, 1987.

[6] Kelley, J. E. *The Critical Path Method: resources planning and scheduling, in J.F.Muth and G.L.Thompson (eds.).* Prentice-Hall, Englewood Cliffs, NJ,, 1963.

[7] Boctor, F. F. Heuristics for scheduling projects with resource restrictions and several resource-duration modes. *Int. J. of Prodn. Research*, 31(11):2547–2558, 1993.

[8] Yang, K. K., Talbot, F. B., and Patterson, J. H. Scheduling a project to maximize its net present value: an integer programming approach. *European Journal of Operations Research*, 64, 1993.

[9] Leachman, R. C., Dincerler, A., and Kim, S. Resource-constrained scheduling of projects with variable-intensity activities. *IIE Trans.*, 22(1):31–39, 1990.

[10] Sampson, S. E., and Weiss, E. N. Local search techniques for the generalized resource-constrained project scheduling problem. *Naval Research Logistics*, 40, 1993.

[11] Talbot, F. B. Resource-constrained project scheduling with time-resource tradeoffs: the non-preemptive case. *Management Science*, 28, 1982.

[12] Patterson, J. H., Slowinski, R., Talbot, F. B., and Weglarz, J. Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. *European Journal of Operations Research*, 49, 1990.

[13] Slowinski, R. Multiobjective network scheduling with efficient use of renewable and non-renewable resources. *European Journal of Operations Research*, 7, 1981.

[14] Ozdamar, L. and Ulusoy, G. A local constraint based analysis approach to project scheduling under general resource constraints. *European Journal of Operations Research*, 79, 1994.

[15] Dar-El, E. M., and Tur, Y. A multiple resource project scheduling algorithm. *AIIE Trans.*, 9:44-52, 1977.

[16] Dar-El, E. M., and Tur, Y. Resource allocation of a multiresource project for variable resource availabilities. *AIIE Trans.*, 9:299–306, 1978.

[17] Kurtlus, I., and Davis, E. W. Multiproject scheduling categorization of heuristic rule performance. *Management Science*. 28:161–172, 1982.

[18] Mohanty, R. P., and Siddiq, M. K. Multiple projects - multiple resource-constrained scheduling: some studies. *Int. J. of Prodn. Research*, 27(2):261–280, 1989.

[19] Moodie, C. L. and Mandeville, D.E. Project resource balancing by assembly line balancing techniques. *Journal of Industrial Engg.*, 17, 1966.

[20] Davis, E. W. Project scheduling under resource constraints: historical review and categorization of procedures. *AIIE Trans.*, 5(4):297–313, 1978.

[21] Andijani, A. A. and Moizuddin, M. Assembly-line balancing: The state-of-the-art review. *Working paper, KFUPM, Dhahran, SA*, 1996.

[22] Pritsker, A. A., Watters, L. J. and Wolfe, P. M. Multi project scheduling with limited resources: a 0-1 programming approach. *Management Science*, 16, 1969.

[23] Herroelen, W. S. Resource-constrained project scheduling: The state-of-the-art. *J. of Operational Research Society*, 23:261-275, 1972.

[24] Icemeli, O., Erengul, S. S., and Zappe, C. J. Project scheduling problems: a survey. *Journal of Operations and Production Management*, 13:80–91, 1993.

[25] Alvarez-Valdes, A. and Tamarit, J. M. *Heuristic algorithm for resource constrained project scheduling: a review and empirical analysis.* in Advances in Project Scheduling, Elsevier, Amsterdam., 1989.

[26] Baroum, S. and Patterson, J.H. A heuristic algorithm for maximizing the net present value of cash flows in resource constrained project schedule. *Working paper, Indiana University*, 1989.

[27] Bell, C.E. and Park,K. Solving resource constrained project scheduling problems by A* search. *Naval Research Logistics*, 37:61–84, 1990.

[28] Doersch, R. H., and Patterson, J. H. Scheduling a project to maximize its present value: a zero-one programming approach. *Management Science*, 23:882–889, 1977.

[29] Bell, C.E. and Han, J. A new heuristic solution method in resource constrained project scheduling problems. *Naval Research Logistics*, 38:315–331, 1991.

[30] Elmaghraby, S. E. and Herroelen, W. S. Scheduling of activities to maxmize npv of the projects. *European J. of Operations Research*, 49:35–40, 1990.

[31] Boctor, F. F. Some efficient multi-heuristic procedures for resource constrained project scheduling. *European Journal of Operations Research*, 49:3–13, 1990.

[32] Russell, A. H. Cash flows in networks. *Management Science*, 16:357-373, 1970.

[33] Russell, A. H. A comparison of heuristics for scheduling projects with cash flows and resource restrictions. *Management Science*, 32:1291-1300, 1986.

[34] Christofides,N. and Alvarez-Valdes,R. and Tamarit,J.M. Project scheduling with resource constraints: a branch and bound approach. *European Journal of Operational Research*, 29:262-273, 1987.

[35] Smith-Daniels, D. E. and Aquilano, N. J. Using a late start resource constrained project schedule to improve project npv. *Decision Sciences*, 18:617-630, 1987.

[36] Cooper, D. F. Heuristics for scheduling resource constrained scheduling projects: an experimental investigation. *Management Science*, 22:1186-1194, 1976.

[37] Davis, E. W., and Heidorn, G. E. An algorithm for optimal project scheduling under multiple resource constraints. *Management Science*, 17:803-816, 1971.

[38] Davis, E. W., and Patterson, J. H. A comparison of heuristics and optimum solutions in resource constrained project scheduling. *Management Science*, 21:944-955, 1975.

[39] Khattab,M. and Choobineh, F. A new heuristic for project scheduling with a single resource constraint. *Comp. Ind.Eng*, 19:514-518, 1990.

[40] Khattab, M. M. and Choobineh, F. A new approach for project scheduling with a limited resource. *Int. J. of Prodn. Research*, 29(1):185-198, 1991.

[41] Patterson,J.H. and Huber, W.D. A horizon-varying, zero - one approach to project scheduling. *Management Science*, 20:990–998, 1974.

[42] Patterson,J.H. and Roth, G.W. Scheduling a project under multiple resource constraints: a 0-1 programming approach. *AIEE Transactions*, 8:449–455, 1976.

[43] Shrage, L. Solving resource constrained network problems by implicit enumeration - nonpreemptive case. *Operations Research*, 18:263–278, 1970.

[44] Stinson, J.P. and Davis, E.W. and Khumawala, B.W. Multiple resource constrained scheduling using branch and bound. *AIEE Transactions*, 10:252–259, 1978.

[45] Talbot, F.B. and Patterson, J.H. An efficient integer programming algorithm with network cuts for solving resource constrained scheduling problems. *Management Science*, 24:1163–1174, 1978.

[46] Talbot, F.B. Resource constrained project scheduling with time-resource tradeoffs: the nonpreemptive case. *Management Science*, 28:1197–1210, 1982.

[47] Thesen,A. Heuristic scheduling of activities under resource and precedence restrictions. *Management Science*, 23:412–422, 1976.

[48] Weglarz, J. Control in resource allocation systems. *Foundations Control Engineering*, 5:159–180, 1980.

[49] Willis, R. J. and Hastings, N.A.J. Project scheduling with resource constraints using branch and bound methods. *Journal of the Operational Research Society*, 27:341–349, 1976.

[50] Tavares, L. V. A multi-stage non-deterministic model for project scheduling under resource constraints. *European J. of Operations Research*, 49:92–101, 1990.

[51] T. J. R. Johnson. *An algorithm for the resource constrained project scheduling problem.* PhD thesis, MIT, 1967.

[52] Patterson,J.H. A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Management Science*, 30:854–867, 1984.

[53] Hastings, N.A.J. On resource allocation in project networks. *Journal of the Operational Research Society*, 23:217–221, 1972.

[54] Shrange, L. Solving resource constrained network problems by implicit enumeration - non preemptive case. *Operations research*, 18:263–278, 1970.

[55] Demeulemeester, E. and Herroelen, W. A branch and bound procedure for the multiple resource constrained project scheduling problem. *Management Science*, 38:1803–1818, 1992.

[56] Wiest, J.D. Some properties of schedules for large projects with limited resources. *Operations Research*, 12:395–418, 1964.

[57] Thesen,A. Measures of restrictiveness of project networks. *Networks*, 7:193–208, 1977.

[58] Patterson,J.H. Project scheduling: the effects of problem structure on heuristic performance. *Naval Research Logistics Quaterly*, 23:95–123, 1976.

[59] Ulusoy, G. and Ozdamar,L. Heuristic performance and network/resource characteristics in resource-constrained project scheduling. *Journal of the Operational Research Society*, 40:1145–1152. 1989.

[60] Ulusoy, G. and Ozdamar, L. . A constraint based perspective in resource constrained project scheduling. *International Journal of Production Research*, 32:693–705, 1994.

[61] Li, K.Y. and Willis, R.J. An iterative scheduling technique for resource-constrained project scheduling. *European Journal of Operational Research*, 56:370–379, 1992.

[62] Cheng, R. and Gen, M. Resource constrianed project scheduling problem using genetic algorithms. *submitted to Int. J. of Intelligent Automation & Soft Computing*, 1995.

[63] Cheng, R. and Gen, M. Evolution program for resource-constrained project scheduling problem. In *The Proceedings of the IEEE Conference on Evolutionary Computation*. IEEE press, 1994.

[64] Ahuja, R. K., Magananti, T. L., and Orlin, J. B. *Network Flow Theory, Algorithms and Applications*. Prentice Hall. 1993.

[65] Kirkpatrick, S., Gelatt, Jr., and Vecchi, M. P. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[66] Cerny, V. Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *J. of Optimization Theory and Applications*, 45:41–51, 1985.

[67] Selim, S. Z., and Al-Sultan, K. S. A simulated annealing algorithm for the clustering problem. *Pattern Recognition*, 24, 1993.

[68] Al-Sultan, K. S., and Selim, S. Z. A global algorithm for the fuzzy clustering problem. *Pattern Recognition*, 26(2):1357-1361, 1993.

[69] Otten, R. H. J. M., and van Ginnekan, L. P. P. P. *The Annealing Algorithm.* Kluwer Academic Publishers, U. S. A., 1989.

[70] Eglese, R. W. Simulated annealing: a tool for operational research. *European J. of O. R.*, 46:271–281, 1978.

[71] Koulamas, C., Antony, S. R. and Jaen, R. A survey of simulated annealing applications to operations research problems. *Omega, Int. J. Mgmt. Sci.*, 22(1):41–56, 1994.

[72] Sait, S. M. and Youssef, H. *VLSI Physical Design Automation: Theory and Practice.* McGraw-Hill Book Co., Europe, 1995.

[73] Sait, S. M. and Youssef, H. *Iterative Computers Algorithms and Their Applications in Engineering.* IEEE Computer Society Press (to be published), 1996.

[74] Laarhoven, P. J. V. and Aarts E. H. L. *Simulated Annealing: Theory and Applications.* Reidel, Deordrecht, 1987.

[75] Holland, J. Genetic algorithms and the optimal allocation of trials. *SIAM J. Comput.*, 2(2):81–91, 1973.

[76] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading MA, 1989.

[77] Biegel, J. E., and Davern, J. J. Genetic algorithms and job-shop scheduling. *Computers and I. E.*, 19:81–91, 1990.

[78] Davis, L. (ed). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[79] Koza, J. R. *Genetic Programming II - Automatic Discovery of Reusable Programs.* MIT Press, U. S. A., 1994.

[80] Grefenstette, J. J., and Baker, J. E. How genetic algorithms work: A critical look at implicit parallelism. In *The Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, 1989.

[81] Kumar, A., Pathak, R. M., Gupta, Y. P., and Parsaei, H. R. A genetic algorithm for distributed system topology design. *Computers and Indistrial Engineering*, 28(3):659–670, 1995.

[82] Kornmal, R. A. and Peterson, A. V. On the alias method for generating random variables from discrete distribution. *American Statistician*, 33:214–218, 1979.

[83] Law, A. M, and Kelton, W. D. *Simulation Modelling & Analysis.* McGraw-Hill International Editions, U. S. A., 1991.

[84] Glover, F. Heuristic for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[85] Glover, F. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533-549, 1986.

[86] Glover, F. and Greenberg, H. J. New approaches for heuristic search: a bilateral linkage with artificial intelligence. *European J. of Operations Research*, 39:119-130, 1989.

[87] Glover, F., Taillard, E. and Werra, D. A user's guide to tabu search. *Annals of Operations Research*, 41:3-28, 1993.

[88] Al-Sultan, K. S. A tabu search approach to the clustering problem. *Pattern Recognition*, 28(9):1443-1451, 1995.

[89] Al-Fawzan, M. and Al-Sultan, K. S. A tabu search algorithm for minimizing the makespan in job shop scheduling. In *The 5th Industrial Engineering Research Conference*. Minneapolis, Minnesota, 1996.

[90] Al-Fawzan, M. and Al-Sultan, K. S. Constrained global optimization by tabu search. In *The 20th International Conference on Computers and Industrial Engineering*. Kyongju, Korea, 1996.

[91] Glover, F. Tabu search - part i. *ORSA J. of Computing*, 1:190-206, 1990.

[92] Lundy, M. and Mees, A. Convergence of an annealing algorithm. *Mathematical Programming*, 34:111-124, 1986.

[93] Montgomery, D. C. *Design and Analysis of Experiments*. John Wiley & Sons, NY, 1991.

[94] Myers, R. H., Khuri, A. I., and Carter, W. H. Response surface methodology: 1966-1988. *Technometrics*, 31(2):137–157, 1989.

Vita

Mohammed Moizuddin

Born in Hyderabad, India.

Received Bachelor of Engineering (B.E) degree in Production Engineering from Osmania University, Hyderabad, India (July, 1993).

Received Master of Science degree in Systems Engineering from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia (June 1996).

Joined the department of Systems Engineering at King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, as a Research Assistant in January, 1994.

Field of Interest include Project Scheduling, Applied Optimization, Simulation and Quality Control

Hobbies include playing cricket, and reading novels.